Microsoft® QuickBASIC

Programmare in BASIC

Microsoft®

Microsoft® QuickBASIC
Programmare in BASIC

Versione 4.5

Per personal computer IBM® e compatibili

Microsoft Corporation

Panoramica

Indice

5 Grafica

6 Gestione degli errori e degli eventi

7 Programmazione a moduli

Parte seconda: Gli elementi essenziali del BASIC

Appendice H Creazione ed utilizzo delle librerie Quick

Appendice I Messaggi di errore

Indice analitico

Introduzione

La 4.5 è una versione sofisticata di Microsoft® QuickBASIC, che potenzia il BASIC e allo stesso tempo ne facilita l'utilizzazione. Essa fornisce il BASIC più avanzato che sia tuttora offerto su microcomputer, agevolato da un ambiente operativo che permette all'utente di concentrarsi sulla creazione di un programma – e non sui meccanismi di struttura e di messa a punto.

Il linguaggio QuickBASIC

Il programmatore in BASICA (o in una simile versione interpretata di BASIC), si renderà conto di quanto le caratteristiche del linguaggio evoluto QuickBASIC semplifichino la scrittura e la manutenzione del proprio software. Per esempio:

- L'istruzione **SELECT CASE** semplifica il trasferimento del controllo a qualunque blocco di codice senza bisogno di nidificare istruzioni **IF...THEN...ELSE**. Tale istruzione offre un'ampia scelta di espressioni prova, per poter creare esattamente la comparazione necessaria.
- Le procedure **SUB** e **FUNCTION** del linguaggio QuickBASIC permettono di disporre in sottoprogrammi gruppi di istruzioni che il programma principale può richiamare ripetutamente. La modularità di QuickBASIC facilita la memorizzazione di tali procedure e fa in modo che esse possano essere riutilizzate in altri programmi.
- Le procedure QuickBASIC sono completamente ricorsive – una procedura può autochiamarsi ripetutamente: ciò semplifica la programmazione di molti algoritmi numerici e di ordinamento che si esprimono meglio in modo ricorsivo.

- Si possono definire tipi di dati composti da una combinazione di variabili intere, reali e di stringa. Variabili connesse tra loro possono essere convenientemente raggruppate sotto un unico nome; ciò facilita il loro passaggio ad una procedura o il loro inserimento in un file.
- QuickBASIC agevola l'accesso ai file binari. I programmi possono leggere e manipolare file di qualsiasi formato, perchè con I/O binario si può accedere direttamente a qualsiasi byte del file.

QuickBASIC è un efficace strumento di sviluppo per uso professionale. Tuttavia, è anche il linguaggio ideale per i programmatori alle prime armi e per quelli di un certo livello che, pur non essendo degli esperti, necessitano di un linguaggio che li aiuti a programmare in maniera efficiente.

L'ambiente QuickBASIC

Oltre ad essere un linguaggio evoluto, il QuickBASIC è anche un ambiente integrato di programmazione che semplifica notevolmente la scrittura e la messa a punto del software.

- Durante la scrittura del programma un intelligente editor di testo controlla gli errori di sintassi di ogni riga. Premendo un solo tasto, il programma entra immediatamente in esecuzione. In caso di errore, si può utilizzare l'editor a tutto schermo per correggerlo, e poi eseguirlo nuovamente.
- Si può effettuare la messa a punto dei programmi senza uscire da QuickBASIC. Il debugger interno permette di esaminare e modificare le variabili, di eseguire qualunque parte del programma o di arrestarne l'esecuzione in particolari condizioni. Tutto ciò avviene all'interno dell'ambiente QuickBASIC. Non si deve modificare il programma o aggiungere l'istruzione **PRINT**.
- La versione 4.5 QuickBASIC contiene due nuovi comandi per un ulteriore potenziamento della messa a punto: **Osservazione immediata** e **Interrompi su errore**.
- Altrettanto nuovo per la versione 4.5 è il Microsoft Consulente QB – la guida in linea. Il Consulente QB è sempre a portata di mano, sia durante la scrittura e l'esecuzione, che la messa a punto. Dopo aver semplicemente posizionato il cursore sulla parola chiave o sul nome definito dall'utente, sul quale si desiderano maggiori informazioni, si preme il tasto F1. Il Consulente QB descrive la sintassi delle istruzioni e delle funzioni del BASIC, ne spiega l'uso e fornisce perfino utili esempi di programmazione.

Come utilizzare il manuale

Il presente manuale si divide in tre parti. La prima, "Elementi di programmazione", fornisce alcune informazioni su tecniche e strategie di programmazione specifiche; la seconda, "Gli elementi essenziali del BASIC", e le appendici, offrono importante materiale di riferimento.

Elementi di programmazione

Ciascun capitolo di questa prima sezione è dedicato ad una singola area di programmazione. Lo studio di questo materiale aiuterà l'utente a familiarizzarsi velocemente con:

- Strutture di controllo
- Procedure **SUB** e **FUNCTION**
- I/O su file e su periferica
- Manipolazione di stringhe
- Grafica
- Gestione degli errori e degli eventi
- Programmazione a moduli

Ogni argomento viene presentato in maniera lineare e facilmente comprensibile, e brevi esempi di programmazione descrivono il funzionamento di ogni parte del BASIC.

Gli argomenti sono presentati in sequenza, dai più semplici ai più complessi; ciò permette di lavorare tranquillamente su un materiale già disposto in ordine logico. L'accento all'interno di questa sezione è sulle applicazioni di QuickBASIC per la risoluzione di problemi pratici, più che sulla teoria.

Oltre ai piccoli esempi, ciascun capitolo contiene programmi completi e funzionali relativi ai principi di programmazione presentati. Per maggior comodità, questi programmi sono inclusi nei dischetti di QuickBASIC.

L'esperto programmatore sceglierà dall'indice gli argomenti di maggior interesse; quello un po' meno esperto, invece, leggerà ciascun capitolo dall'inizio alla fine. Chi non ha mai programmato in alcun linguaggio inizierà dal quarto capitolo del manuale *L'ambiente di programmazione Microsoft QuickBASIC*, "Intermezzo: introduzione al BASIC".

Indipendentemente dagli interessi e dall'esperienza personale, i seguenti sette capitoli forniranno le indicazioni necessarie per imparare a scrivere sofisticate applicazioni in BASIC.

Gli elementi essenziali del BASIC

La seconda parte del manuale, "Gli elementi essenziali del BASIC", si articola in due sezioni ed è un'utile guida di riferimento alle funzioni ed istruzioni del BASIC.

Il capitolo 8, "Panoramica delle istruzioni e funzioni", è un indice in ordine alfabetico di ogni parola chiave del BASIC, descritta nel suo funzionamento o uso e nella sua sintassi. Sarà utile consultare questa sezione del manuale se occorrono delucidazioni riguardo alle istruzioni e alle funzioni.

Nel capitolo 9, "Tabelle di riferimento rapido", le istruzioni e le funzioni del BASIC più comunemente utilizzate sono elencate in sei sezioni sotto forma di tabella; di ciascuna istruzione viene fatta una breve descrizione. I contenuti di questa sezione riprendono gli argomenti presentati nei primi sei capitoli, "Elementi di programmazione". Per informazioni su un particolare processo di programmazione, consultare questa parte del manuale.

Appendici

La terza sezione di questo manuale consiste in un gruppo di appendici contenenti delle informazioni relative a:

- Conversione dei programmi BASICA in QuickBASIC
- Differenze dalle versioni precedenti di QuickBASIC
- Limiti in QuickBASIC
- Codici della tastiera e codici dei caratteri ASCII
- Parole riservate del BASIC
- Metacomandi
- Compilazione ed esecuzione del link da DOS
- Creazione ed utilizzo delle librerie Quick
- Messaggi di errore

Convenzioni tipografiche

Questo manuale fa uso delle seguenti convenzioni tipografiche:

Esempio di convenzione	*Descrizione*
QB.LIB, ADD.EXE, COPY, LINK, /X	Le lettere maiuscole indicano i nomi dei file ed i comandi DOS; sono anche usate per opzioni della linea di comando (a meno che l'applicazione accetti solo i caratteri minuscoli).
SUB, **IF**, **LOOP**, **PRINT**, **WHILE**, **TIME$**	Le lettere maiuscole in grassetto indicano specifiche parole chiave del linguaggio, con significati precisi nel Microsoft BASIC. Le parole chiave sono una parte indispensabile della sintassi di una istruzione, a meno che non siano racchiuse tra parentesi quadre, come spiegato in seguito. L'immissione di parole chiave nel programma deve seguire esattamente le indicazioni; generalmente però è permesso usare sia lettere maiuscole che minuscole.
Ricerca, **Sostituisci**, **Parola intera**, **Tutti i moduli**, **Guida**	Le parole o frasi in grassetto con iniziale maiuscola indicano i nomi e i comandi dei menu e le opzioni e i pulsanti di comando delle finestre di dialogo dell'ambiente QuickBASIC.
`CALL NuovaProc (arg1!, var2%)`	Questo tipo di carattere si utilizza per esempi di programma, output di programma e messaggi di errore all'interno del testo.

`'$INCLUDE:'BC.BI'`
`.`
`.`
`.`
`CHAIN "Prog1"`
`END`
	Tre punti in colonna indicano che una parte del programma di esempio è stata intenzionalmente omessa.

`' Effettua un` `' passaggio`	L'apostrofo (una sola virgoletta a destra) segna l'inizio di un commento in esempi di programmazione.
specfile	Le parole in corsivo sono segnaposti per informazioni che deve fornire l'utente, come il nome di un file; il corsivo è anche occasionalmente usato per enfatizzare il testo.

continua

Esempio di convenzione	*Descrizione*
[*elem-facolt*]	Le voci all'interno di parentesi quadre sono facoltative.
{scelta1 | scelta2}	Le parentesi graffe ed una barra verticale indicano una scelta tra due o più voci. Si deve scegliere una voce, a meno che non siano tutte anche racchiuse tra parentesi quadre.
elementi ripetuti...	I tre puntini dopo una voce indicano la possibile presenza di ulteriori elementi simili.
ALT+F1	Le parole in maiuscoletto sono usate per i tasti e le sequenze di tasti, come INVIO e CTRL+R. Un più (+) indica una combinazione di tasti. Ad esempio, CTRL+E significa tenere premuto il tasto CTRL mentre si preme il tasto E. Il tasto INVIO, talvolta contrassegnato da una freccia piegata, è indicato con ENTER su alcune tastiere. I tasti di spostamento del cursore (le "frecce") sul tastierino numerico sono chiamati tasti di DIREZIONE. I singoli tasti di DIREZIONE sono indicati dalla direzione della freccia (SINISTRA, DESTRA, SU, GIU') o dal nome (PGSU, PGGIU') sul tasto. I nomi dei tasti usati in questo manuale corrispondono a quelli dei personal computer IBM. Altre tastiere possono utilizzare nomi differenti.
"termine nuovo"	Le virgolette in genere indicano la definizione di un nuovo termine nel testo.
`45678.01`	I numeri nel linguaggio BASIC seguono le convenzioni statunitensi del punto separatore al posto della virgola. *Questo formato è obbligatorio per tutti i programmi BASIC*.
`PRINT "Questa è _` `un'unica riga"`	Il segno "_", che talvolta viene usato negli esempi di programma, indica che la riga successiva non è che la continuazione della stessa riga BASIC. In altri termini, quando si digita l'esempio all'interno dell'editor QuickBASIC, non bisogna andare a capo se si trova il segno "_" alla fine di una riga nel manuale.

La sintassi riportata qui sotto per le istruzioni "**LOCK...UNLOCK**" illustra molte delle convenzioni tipografiche di questo manuale:

LOCK [#] *numerofile* [, {*record* | [*inizio*] **TO** *fine*}]
.
.
.
UNLOCK [#] *numerofile* [, {*record* | [*inizio*] **TO** *fine*}]

Nota All'interno di questo manuale, il termine DOS si riferisce ai sistemi operativi MS-DOS® e IBM PC-DOS. Il nome di uno specifico sistema operativo è usato quando si vogliono sottolineare delle caratteristiche che sono particolari a quel sistema. Il termine BASICA si riferisce a versioni interpretate del BASIC in generale.

Stile di programmazione nel manuale

Le seguenti convenzioni sono state usate nella scrittura dei programmi in questo manuale e sui dischi di distribuzione. Sono semplicemente dei consigli per facilitare la lettura dei programmi, e non è obbligatorio seguirli nella stesura dei propri programmi.

- Le parole chiave e le costanti simboliche sono scritte in lettere maiuscole:

```
' PRINT, DO, LOOP, e UNTIL sono parole chiave:
PRINT "Prima pagina"
DO LOOP UNTIL Risposta$ = "N"

' FALSO e VERO sono costanti simboliche uguali a
' 0 e -1 rispettivamente:
CONST FALSO = 0, VERO = NOT FALSO
```

- I nomi delle variabili sono scritti in lettere minuscole con l'iniziale maiuscola; in quelle con più di una sillaba possono comparire altre lettere maiuscole per chiarirne la scansione:

```
NumRecord% = 45
DataNasc$ = "24/11/54"
```

continua

- Le etichette di riga sono usate al posto dei numeri di riga. L'uso delle etichette di riga è limitato alle routine di rilevamento e gestione di errori ed eventi, e all'istruzione **DATA** riferita da **RESTORE**:

```
' GestTemp e DatiSchermo2 sono etichette di riga:
ON TIMER GOSUB GestTemp
RESTORE DatiSchermo2
```

- Come detto nella sezione precedente, un singolo apostrofo (`'`) introduce i commenti:

```
' Questo è un commento; queste righe vengono ignorate
' durante l'esecuzione del programma
```

- I blocchi di controllo del flusso e le istruzioni in procedure o subroutine sono indentate nel codice:

```
SUB OttInput STATIC
   FOR I% = 1 TO 10
      INPUT X
      IF X > 0 THEN
         .
         .
         .
      ELSE
         .
         .
         .
      END IF
   NEXT I%
END SUB
```

- I numeri nel linguaggio BASIC seguono le convenzioni statunitensi del punto separatore al posto della virgola. *Questo formato è obbligatorio per tutti i programmi BASIC*.

```
45678.01
12.5
.33333
```

Parte prima:
Elementi di programmazione

QB
Parte prima

Elementi di programmazione

Vengono trattati i concetti fondamentali della programmazione in BASIC, iniziando dagli argomenti più semplici.

Nel primo capitolo si parla delle strutture che controllano l'esecuzione del programma (le cosiddette strutture di controllo). Nel secondo vengono introdotte le importanti procedure **SUB** e **FUNCTION**. Il terzo capitolo spiega come utilizzare QuickBASIC per operare con i dati. L'utilizzo delle stringhe di testo è l'argomento del quarto capitolo, mentre nel quinto vengono presentate le capacità grafiche di QuickBASIC.

Negli ultimi due capitoli vengono contemplati argomenti più complessi: il sesto riguarda la gestione degli errori e degli eventi, mentre il settimo spiega i vantaggi della programmazione a moduli.

Capitoli

1 Strutture di controllo

Il presente capitolo spiega come controllare l'ordine di esecuzione del programma utilizzando le strutture di controllo, cioè i cicli e le istruzioni di decisione. I cicli fanno eseguire una sequenza di istruzioni quante volte l'utente lo desideri. Le istruzioni di decisione permettono al programma di scegliere tra diverse alternative.

Al termine di questo capitolo si sapranno eseguire i seguenti processi relativi all'utilizzo dei cicli e delle istruzioni di decisione nei programmi BASIC:

- Confrontare espressioni utilizzando gli operatori relazionali
- Unire espressioni a stringa o numeriche per mezzo di operatori logici e stabilire se l'espressione derivante è vera o falsa
- Creare diramazioni nel flusso di un programma con le istruzioni **IF...THEN...ELSE** e **SELECT CASE**
- Creare cicli che ripetono istruzioni un determinato numero di volte
- Creare cicli che ripetono istruzioni finché non o purché sia valida una certa condizione

1.1 Cambiamento dell'ordine di esecuzione delle istruzioni

In mancanza di istruzioni di controllo, l'esecuzione del programma procede da sinistra a destra e dalla prima all'ultima istruzione. Se per alcuni programmi molto semplici è sufficiente questo flusso monodirezionale, l'efficacia e l'utilità di qualsiasi linguaggio di programmazione deriva per lo più dalla capacità di cambiare l'ordine di esecuzione delle istruzioni mediante le strutture di decisione e i cicli.
Con una struttura di decisione un programma può valutare un'espressione per poi saltare a un particolare gruppo di istruzioni (un "blocco") in base al risultato della valutazione. Con un ciclo, invece, esso può eseguire ripetutamente interi blocchi di istruzioni.

Chi si appresta ad operare con questa versione del BASIC dopo aver programmato in BASICA apprezzerà l'ulteriore versatilità di queste strutture di controllo supplementari:

- Il blocco di istruzioni **IF...THEN...ELSE**
- L'istruzione **SELECT CASE**
- Le istruzioni **DO...LOOP** e **EXIT DO**
- L'istruzione **EXIT FOR**, che offre un'alternativa d'uscita dalle iterazioni **FOR...NEXT**

1.2 Espressioni booleane

Si dice espressione booleana un'espressione con il valore "vero" o "falso". Il BASIC utilizza espressioni booleane in certi tipi di cicli e di strutture di decisione. La seguente istruzione **IF...THEN...ELSE** contiene un'espressione booleana, `X < Y`:

```
IF X < Y THEN CALL Procedura1 ELSE CALL Procedura2
```

Nell'esempio precedente, se l'espressione booleana è vera (se il valore della variabile `X` è realmente minore di quello della variabile `Y`), allora viene eseguita la `Procedura1`; altrimenti, (se il valore di `X` è maggiore o uguale al valore di `Y`) viene eseguita la `Procedura2`.

L'esempio precedente dimostra anche come vengono comunemente utilizzate le espressioni booleane: il confronto tra altre due espressioni (in questo caso, `X` e `Y`) per stabilirne la relazione. Questi confronti vengono fatti con gli operatori relazionali, come mostrato nella tabella 1.1.

Tabella 1.1 Operatori relazionali utilizzati in BASIC

Operatore	***Significato***
=	Uguale a
<>	Diverso da
<	Minore di
<=	Minore o uguale a
>	Maggiore di
>=	Maggiore o uguale a

Si possono utilizzare questi operatori relazionali per confrontare tra loro espressioni a stringa. In questo caso, "maggiore di", "minore di", e così via si riferiscono all'ordine alfabetico. Ad esempio, l'espressione seguente è vera dal momento che la parola "deduce" alfabeticamente precede la parola "deduci":

```
"deduce" < "deduci"
```

Le espressioni booleane utilizzano frequentemente anche gli "operatori logici" **AND**, **OR**, **NOT**, **XOR**, **IMP**, e **EQV**. Questi operatori permettono di comporre espressioni booleane composte. Ad esempio,

espressione1 **AND** *espressione2*

è un'espressione vera solo se *espressione1* e *espressione2* sono entrambe vere. Di conseguenza, nell'esempio che segue, verrà visualizzato il messaggio `"Tutto in ordine"` solo se entrambe le espressioni booleane `X <= Y` e `Y <= Z` sono vere:

```
IF (X <= Y) AND (Y <= Z) THEN PRINT "Tutto in ordine"
```

Le parentesi che racchiudono le espressioni booleane nell'esempio precedente non sono strettamente necessarie, dal momento che gli operatori relazionali come <= vengono valutati prima degli operatori logici come **AND**. Le parentesi facilitano la lettura di espressioni booleane complesse ed assicurano che il loro contenuto venga valutato nell'ordine desiderato.

Il BASIC utilizza i valori numerici −1 e 0 come simboli, rispettivamente, dei valori vero e falso. Si può verificare ciò chiedendo al computer di visualizzare un'espressione vera ed una falsa, come nell'esempio seguente.

```
x = 5
y = 10
Print x < y         ' Valuta e visualizza un'espressione
                    ' booleana "vera".
Print x > y         ' Valuta e visualizza un'espressione
                    ' booleana "falsa".
```

Output

```
-1
 0
```

L'assegnazione di −1 al valore "vero" si può comprendere esaminando come funziona l'operatore BASIC **NOT**. **NOT** inverte ciascun bit nella rappresentazione binaria del suo operando, sostituendo i bit 0 con bit 1, e viceversa. Inoltre, dal momento che il valore intero 0 (falso) viene memorizzato internamente come una sequenza di sedici bit 0, il valore di **NOT** 0 (vero) viene memorizzato in una sequenza di sedici bit 1, come segue:

```
FALSO = 0000000000000000

VERO = NOT FALSO = 1111111111111111
```

Nel metodo del complemento a due che il BASIC utilizza per memorizzare gli interi, sedici bit 1 rappresentano il valore −1. Notare che il BASIC restituisce −1 quando valuta vera un'espressione booleana; però considera vero qualsiasi valore che non sia zero, come si vede dall'output nel seguente esempio:

```
INPUT "Digitare un valore:", x

IF x THEN PRINT x "è vero."
```

Output

```
Digitare un valore: 2
 2 è vero.
```

L'operatore **NOT** in BASIC è un operatore "bit per bit". Alcuni linguaggi di programmazione, come il C ed il Pascal, hanno sia un operatore **NOT** bit per bit sia un operatore **NOT** "logico".

La distinzione è la seguente:

- Un operatore **NOT** bit per bit restituisce il valore di falso (0) solo al valore −1.
- Un operatore **NOT** logico restituisce il valore di falso (0) a qualsiasi valore vero (cioè non uguale a zero).

L'espressione **NOT** in BASIC restituisce ancora un valore vero a qualsiasi *espressione* vera che non sia −1, come indicato nell'elenco successivo:

Valore di espressione	*Valore di NOT espressione*
1	-2
2	-3
-2	1
-1	0

Si noti che **NOT** *espressione* è falso solo se *espressione* ha un valore di −1. Nel definire costanti o variabili booleane da utilizzare nei programmi, usare sempre il valore −1 per indicare vero.

I valori 0 e −1 si possono usare per definire delle utili costanti mnemoniche booleane da utilizzare nei cicli e nelle decisioni. Molti esempi del presente manuale si avvalgono di questo metodo dimostrato nel seguente frammento di programma, che dispone in ordine crescente gli elementi di una matrice chiamata `Tinta`:

```
' Definisce costanti simboliche da utilizzare nel programma:
CONST FALSO = 0, VERO = NOT FALSO
.
.
.
DO
   Scambi = FALSO
   FOR I = 1 TO NumTransaz - 1
      IF Tinta(I) < Tinta(I + 1) THEN
         SWAP Tinta(I), Tinta(I + 1)
         Scambi = VERO
      END IF
   NEXT I
LOOP WHILE Scambi      ' Continua a iterare mentre Scambi
.                      ' è VERO.
.
.
```

1.3 Strutture di decisione

A seconda del valore di un'espressione, le strutture di decisione fanno eseguire al programma una delle seguenti azioni:

- Eseguire una delle diverse istruzioni alternative all'interno della stessa istruzione di decisione
- Saltare ad una parte del programma al di fuori della struttura di decisione

In BASICA, l'unica struttura di decisione disponibile è l'istruzione monoriga **IF...THEN [...ELSE]**. Nella sua forma più semplice, **IF...THEN**, viene valutata l'espressione che segue la parola chiave **IF**. Se l'espressione è vera, il programma esegue le istruzioni che seguono la parola chiave **THEN**; se invece l'espressione è falsa, il programma prosegue con la riga successiva all'istruzione **IF...THEN**. Le righe 50 e 70 del seguente frammento di programma BASICA sono esemplificative di **IF...THEN**:

```
30  INPUT A
40  ' Se A è maggiore di 100, visualizza un messaggio e
45  ' torna alla riga 30; altrimenti, avanza alla riga 60:
50  IF A > 100 THEN PRINT "Troppo grande": GOTO 30
60  ' Se A è uguale a 100, passa alla riga 300; altrimenti,
65  ' avanza alla riga 80:
70  IF A = 100 THEN GOTO 300
80  PRINT A / 100: GOTO 30
.
.
.
```

Con l'aggiunta della clausola **ELSE**, il programma esegue le operazioni successive alla parola chiave **THEN** se l'espressione dopo **IF** è vera, e le operazioni successive alla parola chiave **ELSE** se è falsa. Il seguente brano di programma mostra l'uso della clausola **ELSE** in un'istruzione **IF...THEN...ELSE**:

```
10  INPUT "Parola d'ordine"; ParOrd$
15  ' Se l'utente digita "spada", salta alla riga 50;
20  ' altrimenti, visualizza un messaggio e ritorna alla
25  ' riga 10:
30  IF ParOrd$ = "spada" THEN 50 ELSE PRINT "Riprova": _
    GOTO 10
.
.
.
```

L'istruzione **IF...THEN...ELSE** del BASICA va bene per le decisioni semplici, ma dà origine a programmi quasi illegibili per quelle più complesse, specialmente se si pongono tutte le decisioni all'interno dell'istruzione **IF...THEN...ELSE**, oppure se si nidificano più istruzioni **IF...THEN...ELSE** (cioè se si inserisce un'istruzione **IF...THEN...ELSE** dentro un'altra, operazione peraltro lecita). Il successivo brano di programma BASICA mostra quanto sia difficile da seguire un test anche semplice:

```
10  ' Le successive istruzioni nidificate IF...THEN...ELSE
15  ' visualizzano un diverso output per ognuno dei quattro
20  ' casi seguenti:
25  ' 1) A <= 50, B <= 50     3) A > 50, B <= 50
30  ' 2) A <= 50, B > 50      4) A > 50, B > 50
35
40  INPUT A, B
45
50  ' Notare: anche se la riga 80 si estende su varie righe
55  ' di schermo, essa è una sola "riga logica" (tutta
60  ' digitata prima di premere il tasto <INVIO>). BASICA
65  ' esegue sullo schermo il ritorno a capo automatico
70  ' delle righe lunghe.
75
80  IF A <= 50 THEN IF B <= 50 THEN PRINT "A <= 50, B <= 50"
ELSE PRINT "A <= 50, B > 50" ELSE IF B <= 50 THEN PRINT "A >
50, B <= 50" ELSE PRINT "A > 50, B > 50"
```

Per evitare queste complicazioni, il BASIC ora usa l'istruzione **IF...THEN...ELSE** in forma di blocco: una decisione non è più limitata ad una sola riga logica. L'esempio mostra lo stesso programma BASICA riscritto in modo da utilizzare blocchi **IF...THEN...ELSE**:

```
INPUT A, B
IF A <= 50 THEN
   IF B <= 50 THEN
      PRINT "A <= 50, B <= 50"
   ELSE
      PRINT "A <= 50, B > 50"
   END IF
ELSE
   IF B <= 50 THEN
      PRINT "A > 50, B <= 50"
   ELSE
      PRINT "A > 50, B > 50"
   END IF
END IF
```

Il QuickBASIC fornisce anche l'istruzione **SELECT CASE...END SELECT** (cui si fa riferimento come **SELECT CASE**) per le decisioni strutturate.

Sia il blocco d'istruzioni **IF...THEN...ELSE** che l'istruzione **SELECT CASE** permettono che l'aspetto del programma ne rifletta la logica, evitando di accentrare più istruzioni su di una stessa riga. Il programmatore potrà operare con maggiore flessibilità, rendendo il proprio programma più leggibile e curandone più facilmente la manutenzione.

1.3.1 Il blocco IF...THEN...ELSE

La tabella 1.2 mostra la sintassi del blocco di istruzioni **IF...THEN...ELSE** e fornisce un esempio della sua utilizzazione.

Tabella 1.2 Sintassi ed esempio del blocco IF...THEN...ELSE

Sintassi	***Esempio***
IF *condizione1* **THEN**	`IF X > 0 THEN`
[*bloccoistruzioni-1*]	`  PRINT "X è positivo"`
[**ELSEIF** *condizione2* **THEN**	`  NumPos = NumPos + 1`
[*bloccoistruzioni-2*]]	`ELSEIF X < 0 THEN`
.	`  PRINT "X è negativo"`
.	`  NumNeg = NumNeg + 1`
.	`ELSE`
[**ELSE**	`  PRINT "X è zero"`
[*bloccoistruzioni-n*]]	`END IF`
END IF	

Gli argomenti *condizione1*, *condizione2* e così via, sono espressioni. Possono essere espressioni numeriche, e allora qualsiasi valore che non sia zero è vero, mentre zero è falso; o espressioni booleane, e allora −1 è vero, mentre zero è falso. Come spiegato nella sezione 1.2, le espressioni booleane generalmente confrontano due espressioni numeriche o a stringa utilizzando uno degli operatori relazionali, quali < o >=.

A ciascuna clausola **IF**, **ELSEIF**, ed **ELSE** fa seguito un blocco di istruzioni. Nessuna istruzione del blocco deve trovarsi sulla stessa riga delle clausole **IF**, **ELSEIF** o **ELSE**, altrimenti il BASIC considera **IF...THEN...ELSE** un'istruzione monoriga.

Il BASIC valuta dall'alto in basso ciascuna espressione nelle clausole **IF** e **ELSEIF**, saltando blocchi di istruzioni finché non trova la prima espressione vera. A questo punto, esegue le istruzioni corrispondenti all'espressione e salta fuori dal blocco, all'istruzione che segue la clausola **END IF**.

Se nessuna delle espressioni nelle clausole **IF** o **ELSEIF** è vera, il BASIC salta alla clausola **ELSE**, se ne esiste una, e ne esegue le istruzioni. Altrimenti, se non esiste una clausola **ELSE**, il programma esegue l'istruzione successiva alla clausola **END IF**.

Entrambe le clausole **ELSE** ed **ELSEIF** sono opzionali, come dimostra l'esempio seguente:

```
' Se il valore di X è minore di 100, esegue le due
' istruzioni che precedono END IF; altrimenti passa
' all'istruzione INPUT che segue END IF:

IF X < 100 THEN
   PRINT X  Numero = Numero + 1
   END IF
INPUT "Valore nuovo"; Risposta$
.
.
.
```

Un solo blocco **IF...THEN...ELSE** può contenere diverse istruzioni **ELSEIF**, come segue:

```
IF c$ >= "A" AND C$ <= "Z" THEN
   PRINT "Maiuscola"
ELSEIF C$ >= "a" AND C$ <= "z" THEN
   PRINT "Minuscola"
ELSEIF C$ >= "=" AND C$ <= ")" THEN
   PRINT "Cifra"
ELSE
   PRINT "Carattere non alfanumerico"
END IF
```

Anche se più di una condizione è vera, viene eseguito al massimo un solo blocco d'istruzioni. Ad esempio, se si digita la parola `asso` come input nell'esempio successivo, esso visualizzerà il messaggio `Input troppo breve` ma non il messaggio `Non deve iniziare per a`:

```
INPUT Verifica$
IF LEN(Verifica$) > 6 THEN
   PRINT "Input troppo lungo"
ELSEIF LEN(Verifica$) < 6 THEN
   PRINT "Input troppo breve"
ELSEIF LEFT$(Verifica$, 1) = "a" THEN
   PRINT "Non deve iniziare per a"
END IF
```

Le istruzioni **IF...THEN...ELSE** possono essere nidificate; in altre parole, si può collocare un'istruzione **IF...THEN...ELSE** all'interno di un'altra istruzione **IF...THEN...ELSE**, come segue:

```
IF X > 0 THEN
   IF Y > 0 THEN
      IF Z > 0 THEN
         PRINT "Tutti maggiori di zero."
      ELSE
         PRINT "Solo X e Y maggiori di zero."
      END IF
   END IF
ELSEIF X = 0 THEN
   IF Y = 0 THEN
      IF Z = 0 THEN
         PRINT "Tutti uguali a zero."
      ELSE
         PRINT "Solo X e Y uguali a zero."
      END IF
   END IF
ELSE
   PRINT "X è minore di zero."
END IF
```

1.3.2 SELECT CASE

L'istruzione **SELECT CASE** è una struttura di decisione a scelte multiple simile all'istruzione a blocchi **IF...THEN...ELSE**. Un blocco **IF...THEN...ELSE** può essere utilizzato ovunque si può utilizzare **SELECT CASE**.

La differenza principale tra le due istruzioni è che **SELECT CASE** esegue istruzioni differenti o salta a diverse parti del programma in base alla valutazione di una singola espressione, mentre un blocco **IF...THEN...ELSE** può valutare espressioni completamente differenti.

Esempi

Gli esempi seguenti illustrano similitudini e differenze tra le istruzioni **SELECT CASE** e **IF...THEN...ELSE**. Ecco un esempio di utilizzo del blocco **IF...THEN...ELSE** per una decisione a scelte multiple.

```
INPUT X
IF X = 1 THEN
   PRINT "uno"
ELSEIF X = 2 THEN
   PRINT "due"
ELSEIF X = 3 THEN
   PRINT "tre"
ELSE
   PRINT "Deve essere un intero compreso tra 1 e 3."
END IF
```

La decisione precedente viene ora riscritta utilizzando **SELECT CASE**:

```
INPUT X
SELECT CASE X
   CASE 1
      PRINT "uno"
   CASE 2
      PRINT "due"
   CASE 3
      PRINT "tre"
   CASE ELSE
      PRINT "Deve essere un intero compreso tra 1 e 3."
END SELECT
```

La decisione seguente può essere presa sia con un'istruzione **SELECT CASE**, sia con un blocco **IF...THEN...ELSE**. L'istruzione **IF...THEN...ELSE** è però più efficiente, perché nelle clausole **IF** ed **ELSEIF** vengono valutate espressioni differenti.

```
INPUT X,Y
IF X = 0 AND Y = 0 THEN
   PRINT "Sono entrambe zero."
ELSEIF X = 0 THEN
   PRINT "Solo X è zero."
ELSEIF Y = 0 THEN
   PRINT "Solo Y è zero."
ELSE
   PRINT "Nessuna delle due è zero."
END IF
```

1.3.2.1 Utilizzo dell'istruzione SELECT CASE

La tabella 1.3 riporta la sintassi di un'istruzione **SELECT CASE** ed un esempio.

Tabella 1.3 Sintassi ed esempio di SELECT CASE

Sintassi

```
SELECT CASE espressione
 CASE elencoespressioni1
  [bloccoistruzioni-1]
 [CASE elencoespressioni2
  [bloccoistruzioni-2]]
 .
 .
 .
 [CASE ELSE
  [bloccoistruzioni-n]]
END SELECT
```

Esempio

```
Input ValoreTest
SELECT CASE ValoreTest
  CASE 1,3,5,7,9
    PRINT "Dispari"
  CASE 2,4,6,8
    PRINT "Pari"
  CASE IS < 1
    PRINT "Troppo basso"
  CASE IS > 9
    PRINT "Troppo alto"
  CASE ELSE
    PRINT "Non un intero"
END SELECT
```

Gli argomenti *elencoespressioni* successivi ad una clausola **CASE** possono essere uno o più dei seguenti, suddivisi da virgole:

- Un'espressione numerica o un intervallo di espressioni numeriche
- Un'espressione a stringa o un intervallo di espressioni a stringa

Per specificare un intervallo di espressioni, usare la sintassi seguente per l'istruzione **CASE**:

CASE *espressione* **TO** *espressione*
CASE IS *operatore-relazionale espressione*

L'*operatore-relazionale* è uno degli operatori elencati nella tabella 1.1. Ad esempio, se si utilizza `CASE 1 TO 4`, le istruzioni relative a questo caso vengono eseguite se l'*espressione* nell'istruzione **SELECT CASE** è maggiore o uguale a 1 e minore o uguale a 4. Se si utilizza `CASE IS < 5`, le relative istruzioni vengono eseguite solo se l'*espressione* è minore di 5.

Nell'esprimere un intervallo con la parola chiave **TO**, mettere prima il valore minore. Ad esempio, per rilevare un valore compreso tra −5 e −1, digitare l'istruzione **CASE** in questo modo:

```
CASE -5 TO -1
```

L'istruzione successiva non è valida per specificare l'intervallo tra −1 e −5: le istruzioni relative a questo caso non verranno mai eseguite:

```
CASE -1 TO -5
```

Analogamente, un intervallo di costanti a stringa va elencato in ordine alfabetico:

```
CASE "oritteropo" TO "orso"
```

Per ciascuna clausola **CASE**, possono essere elencate più espressioni o intervalli di espressioni, come nelle righe seguenti:

```
CASE 1 to 4, 7 TO 9, Jolly1%, Jolly2%
CASE IS = Test$, IS = "fine dei dati"
CASE IS < LimInf, 5, 6, 12, IS > LimSup
CASE IS < "HAN", "MAO" TO "TAO"
```

Se il valore dell'espressione **SELECT CASE** appare nell'elenco che segue una clausola **CASE**, vengono eseguite solo le istruzioni relative a quella clausola. Il controllo poi passa alla prima istruzione eseguibile successiva ad **END SELECT**, piuttosto che al successivo blocco di istruzioni all'interno della struttura **SELECT CASE**, come si vede dall'esempio seguente (ove l'utente digita 1):

```
INPUT X
SELECT CASE X
   CASE 1
      PRINT "Uno, ";
   CASE 2
      PRINT "Due, ";
   CASE 3
      PRINT "Tre, ";
   END SELECT
      PRINT "è tutto"
```

Output

```
? 1
Uno, è tutto
```

Se uno stesso valore o intervallo di valori appare in più di una clausola **CASE**, vengono eseguite solo le istruzioni relative alla prima istanza, come mostra l'output nell'esempio successivo (in cui l'utente ha digitato `INDIGENO`):

```
INPUT Test$
SELECT CASE Test$
   CASE "I" TO "IZZZZZZZZZZZZZZZZZ"
      PRINT "Una parola in maiuscolo con iniziale I"
   CASE IS < "A"
      PRINT "Una sequenza di caratteri non alfabetici"
   CASE "INDIGENO"
      ' Questo caso non viene mai eseguito perché
      ' INDIGENO si trova nell'intervallo della prima
      ' clausola CASE:
      PRINT "Un caso speciale"
END SELECT
```

Output

```
? INDIGENO
Una parola in maiuscolo con iniziale I
```

Se si utilizza una clausola **CASE ELSE**, questa deve essere l'ultima clausola elencata nell'istruzione **SELECT CASE**. Le istruzioni comprese tra la clausola **CASE ELSE** e la clausola **END SELECT** vengono eseguite solo se *espressione* non corrisponde ad alcuna altra selezione di **CASE** nell'istruzione **SELECT CASE**. Inserendo un'istruzione **CASE ELSE** nel blocco **SELECT CASE**, si possono meglio gestire i valori imprevisti di *espressione*. Se non esiste un'istruzione **CASE ELSE** né viene rilevata alcuna corrispondenza per *espressione* in alcuna istruzione **CASE**, il programma continua l'esecuzione.

Esempio

Il seguente brano di programma illustra un comune uso dell'istruzione **SELECT CASE**: esso visualizza un menu sullo schermo e passa a diversi sottoprogrammi in base al numero digitato dall'utente.

```
DO                    ' Inizia il ciclo del menu.

   CLS                ' Cancella lo schermo.
```

```
   ' Visualizza cinque scelte di menu sullo schermo:
   PRINT "Menu principale" : PRINT
   PRINT "1)  Aggiungere nomi nuovi"
   PRINT "2)  Cancellare dei nomi"
   PRINT "3)  Modificare i dati"
   PRINT "4)  Elencare i nomi"
   PRINT
   PRINT "5)  Uscita"

   ' Sollecita l'input:
   PRINT: PRINT "Digitare la scelta (da 1 a 5):"

   ' Attende che l'utente prema un tasto. INPUT$(1)
   ' legge un carattere digitato dalla tastiera:
   Scelta$ = INPUT$(1)

   ' Utilizza SELECT CASE per elaborare la risposta:
   SELECT CASE Scelta$

      CASE "1"
         CALL AggiungiDati
      CASE "2"
         CALL CancellaDati
      CASE "3"
         CALL SostituisciDati
      CASE "4"
         CALL ElencaDati
      CASE "5"
         EXIT DO       ' L'unico modo per uscire dal ciclo
                       ' del menu
      CASE ELSE
         BEEP
   END SELECT

LOOP                   ' Fine del ciclo del menu.

END

' Sottoprogrammi AggiungiDati, CancellaDati,
' SostituisciDati, ed ElencaDati:
.
.
.
```

1.3.2.2 SELECT CASE e ON...GOSUB

E' possibile sostituire la vecchia istruzione **ON** *espressione* **{GOSUB | GOTO}** con l'istruzione **SELECT CASE**, la quale offre sicuramente maggiori modalità d'impiego. L'istruzione **SELECT CASE** ha molti più vantaggi dell'istruzione **ON...GOSUB**, come sintetizzato di seguito:

- Il valore di *espressione* in **SELECT CASE** *espressione* può essere o una stringa o un numero. L'*epressione* nell'istruzione **ON** *espressione* **{GOSUB | GOTO}** deve avere un valore numerico compreso tra 0 e 255.
- L'istruzione **SELECT CASE** salta al blocco d'istruzioni immediatamente successivo alla relativa clausola **CASE**. Al contrario, **ON** *espressione* **GOSUB** salta ad una subroutine in un'altra parte del programma.
- Le clausole **CASE** possono essere utilizzate per verificare un'*espressione* relativa ad un intervallo di valori. Quando l'intervallo è abbastanza ampio, non è facile eseguire il test con **ON** *espressione* **{GOSUB | GOTO}**, come illustrato nei frammenti seguenti.

In questo brano, l'istruzione **ON...GOSUB** passa ad una delle subroutine 50, 100, o 150, in base al valore digitato dall'utente:

```
X% = -1
WHILE X%
   INPUT "Digitare la scelta (o 0 per uscire): ", X%
   IF X% = 0 THEN END
   WHILE X% < 1 OR X% > 12
      PRINT "Il valore deve essere tra 1 e 12"
      INPUT "Digitare la scelta (o 0 per uscire): ", X%
   WEND
   ON X% GOSUB 50,50,50,50,50,50,50,50,100,100,100,150
WEND
.
.
.
```

E' utile ora confrontare l'esempio precedente con il successivo, che utilizza un'istruzione **SELECT CASE** con intervalli di valori in ciascuna clausola **CASE**:

```
DO
   INPUT "Digitare la scelta (o 0 per uscire): ", X%
   SELECT CASE X%
      CASE 0
         END
```

```
      CASE 1 TO 8       ' Rimpiazza la "subroutine 50"
                        ' nell'esempio precedente
      CASE 9 TO 11      ' Rimpiazza la "subroutine 100"
                        ' nell'esempio precedente
      CASE 12           ' Rimpiazza la "subroutine 150"
                        ' nell'esempio precedente
      CASE ELSE         ' L'input era esterno all'intervallo.
         PRINT "Il valore deve essere tra 1 e 12"
      END SELECT
LOOP
```

1.4 Strutture iterative

Le strutture iterative fanno ripetere un blocco di istruzioni (il ciclo) un determinato numero di volte, oppure finché una certa condizione rimane vera o falsa.

Gli utenti di BASICA hanno già dimestichezza con le due strutture iterative **FOR...NEXT** e **WHILE...WEND**, presentate rispettivamente nelle sezioni 1.4.1 e 1.4.2. QuickBASIC offre inoltre anche l'istruzione **DO...LOOP**, descritta nella sezione 1.4.3.

1.4.1 Cicli FOR...NEXT

Un ciclo **FOR...NEXT** fa eseguire un determinato numero di volte le istruzioni contenute nel ciclo, incrementando o decrementando un contatore da un valore iniziale ad un valore finale. Finché il contatore non ha raggiunto il valore finale, il ciclo continua ad eseguire. La tabella 1.4 contiene la sintassi dell'istruzione **FOR...NEXT** ed un esempio di utilizzazione.

Tabella 1.4 Sintassi ed esempio di FOR...NEXT

Sintassi	***Esempio***

FOR *contatore* = *inizio* **TO** *fine* [**STEP** *dimpasso*]
[*bloccoistruzioni-1*]
[**EXIT FOR**
[*bloccoistruzioni-2*]]
NEXT [*contatore*]
	`FOR I% = 1 TO 10`
	`  Matrice%(I%) = I%`
	`NEXT`

In un ciclo **FOR...NEXT**, la variabile *contatore* inizialmente ha il valore dell'espressione *inizio*. Dopo ogni iterazione, il valore di *contatore* viene modificato. Se si omette l'opzione **STEP**, il valore predefinito per tale modifica è uguale ad uno; cioè viene aggiunto uno a *contatore* ogni volta che il ciclo si ripete. Nel caso si utilizzi **STEP**, alla variabile *contatore* viene aggiunto il valore di *dimpasso*. L'argomento *dimpasso* può avere un qualsiasi valore numerico; se esso è negativo, il ciclo esegue un conteggio alla rovescia da *inizio* a *fine*. Dopo che il valore della variabile *contatore* aumenta o diminuisce, viene confrontato con *fine*. A questo punto, se si verifica una delle condizioni seguenti, il ciclo viene terminato:

- Il ciclo sta contando in su (*dimpasso* è positivo) e *contatore* è maggiore di *fine*.
- Il ciclo sta contando alla rovescia (*dimpasso* è negativo) e *contatore* è minore di *fine*.

Nell'illustrazione 1.1 si vede il diagramma logico di un ciclo **FOR...NEXT** quando il valore di *dimpasso* è positivo.

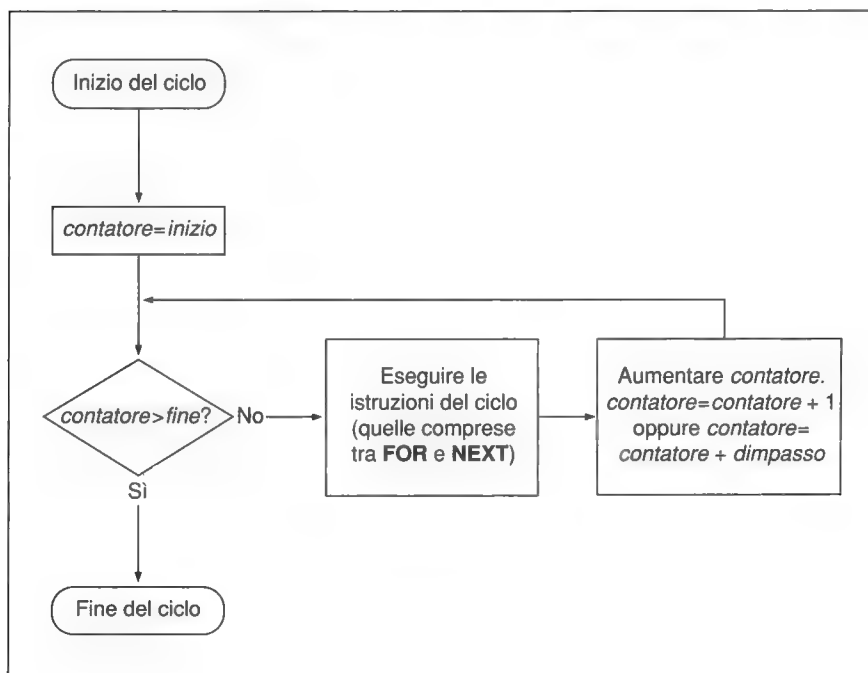
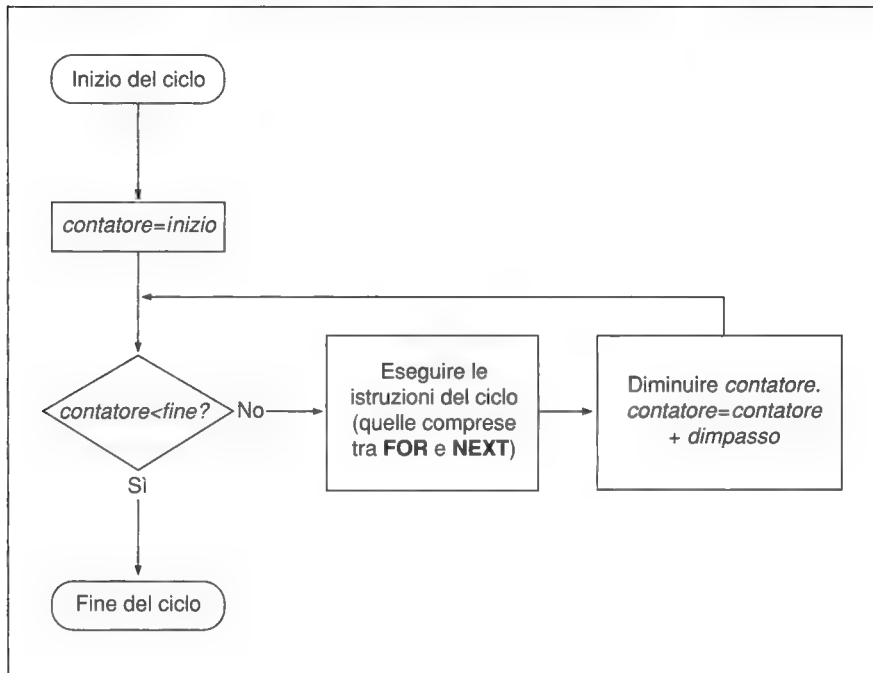
Illustrazione 1.1 Diagramma logico del ciclo FOR...NEXT con STEP positivo

Nell'illustrazione 1.2 si vede il diagramma logico di un ciclo **FOR...NEXT** quando il valore di *dimpasso* è negativo.

Illustrazione 1.2 Diagramma logico del ciclo FOR...NEXT con STEP negativo

Un'istruzione **FOR...NEXT** esegue sempre il "test all'inizio del ciclo", perciò se si verifica una delle due condizioni seguenti, il ciclo non verrà mai eseguito:

- La dimensione del passo è positiva, ma il valore di *inizio* è maggiore del valore di *fine*:

```
' Il ciclo non entra mai in esecuzione, poiché I% inizia
' già maggiore di 9:
FOR I% = 10 TO 9
.
.
.
NEXT I%
```

- La dimensione del passo è negativa, ma il valore di *inizio* è minore del valore di *fine*:

```
' Il ciclo non entra mai in esecuzione, poiché I% inizia
' già minore di -9:
FOR I% = -10 TO -9 STEP -1
.
.
.
NEXT I%
```

Non è necessario utilizzare l'argomento *contatore* nella clausola **NEXT**; però risulta utile per sapere sempre in quale ciclo ci si trova all'interno di diverse iterazioni **FOR...NEXT** nidificate (un ciclo dentro l'altro).

Ecco alcune indicazioni generali per la nidificazione delle iterazioni **FOR...NEXT**:

- Se si utilizza l'argomento *contatore* nella clausola **NEXT**, il contatore di un ciclo interno deve apparire prima dei contatori di ciascun ciclo più esterno. In altre parole, la seguente è una nidificazione lecita:

```
FOR I = 1 TO 10
   FOR J = -5 TO 0
   .
   .
   .
   NEXT J
NEXT I
```

  Però la nidificazione seguente non è lecita:

```
FOR I = 1 TO 10
   FOR J = -5 TO 0
   .
   .
   .
   NEXT I
NEXT J
```

- Se si utilizza una clausola **NEXT** per chiudere ciascun ciclo, il numero delle clausole **NEXT** deve essere uguale a quello delle clausole **FOR**.
- Se si utilizza una singola clausola **NEXT** per terminare contemporaneamente diversi livelli di iterazioni **FOR...NEXT**, le variabili di conteggio devono apparire dopo la clausola **NEXT** nell'ordine "inverso":

  NEXT *contatorepiùinterno*, ... , *contatorepiùesterno*

  In tal caso, il numero delle variabili di conteggio nella clausola **NEXT** deve essere uguale al numero delle clausole **FOR**.

Esempi

I tre brani di programma riportati successivamente illustrano modi diversi di nidificare cicli **FOR...NEXT** aventi tutti lo stesso output come risultato. Il primo esempio mostra cicli **FOR...NEXT** nidificati, con una clausola **NEXT** con la sua variabile di conteggio per ciascun ciclo:

```
FOR I = 1 TO 2
   FOR J = 4 TO 5
      FOR K = 7 TO 8
         PRINT I, J, K
      NEXT K
   NEXT J
NEXT I
```

Anche l'esempio successivo esplicita le variabili di conteggio, ma usa una sola clausola **NEXT** per tutti e tre i cicli:

```
FOR I = 1 TO 2
   FOR J = 4 TO 5
      FOR K = 7 TO 8
         PRINT I, J, K
NEXT K, J, I
```

Nell'ultimo esempio le variabili di conteggio rimangono implicite nelle clausole **NEXT**:

```
FOR I = 1 TO 2
   FOR J = 4 TO 5
      FOR K = 7 TO 8
         PRINT I, J, K
      NEXT
   NEXT
NEXT
```

Output

```
1            4            7
1            4            8
1            5            7
1            5            8
2            4            7
2            4            8
2            5            7
2            5            8
```

1.4.1.1 Uscita da un ciclo FOR...NEXT con EXIT FOR

Si può anche uscire da un ciclo **FOR...NEXT** prima che la variabile di conteggio raggiunga il valore finale del ciclo: basta utilizzare l'istruzione **EXIT FOR**. Un singolo ciclo **FOR...NEXT** può avere un numero qualsiasi di istruzioni **EXIT FOR** e tali istruzioni possono apparire ovunque all'interno del ciclo. L'esempio successivo dimostra un modo per utilizzare l'istruzione **EXIT FOR**:

```
' Visualizza la radice quadrata dei numeri compresi tra 1 e
' 30000. Se l'utente preme un tasto durante l'esecuzione di
' questo ciclo, il ciclo termina:
FOR I% = 1 TO 30000
   PRINT SQR(I%)
   IF INKEY$ <> "" THEN EXIT FOR
NEXT
.
.
.
```

EXIT FOR fa uscire solo dal ciclo **FOR...NEXT** più interno in cui si trova. Ad esempio, se l'utente preme un tasto durante l'esecuzione dei seguenti cicli nidificati, il programma esce semplicemente dal ciclo più interno. Se il ciclo esterno è ancora attivo (cioè se il valore di `I%` <= 100), il controllo torna subito al ciclo interno:

```
FOR I% = 1 TO 100
   FOR J% = 1 TO 100
      PRINT I% / J%
      IF INKEY$ <> "" THEN EXIT FOR
   NEXT J%
NEXT I%
```

1.4.1.2 Arresto dell'esecuzione del programma con FOR...NEXT

Molti programmi BASICA utilizzano un ciclo vuoto **FOR...NEXT**, come il seguente, per arrestare momentaneamente il programma:

```
.
.
' Non ci sono istruzioni all'interno di questo ciclo;
' esso esegue esclusivamente un conteggio da 1 a 10000
' utilizzando degli interi (numeri interi).
FOR I% = 1 TO 10000: NEXT
.
.
.
```

Per creare pause brevi o di durata non precisa, si può tranquillamente utilizzare **FOR...NEXT**. Il problema che scaturisce dall'utilizzazione di un ciclo vuoto **FOR...NEXT** è che computer differenti, differenti versioni di BASIC, o differenti opzioni di compilazione possono influire sui tempi di esecuzione dell'aritmetica all'interno del ciclo stesso. Pertanto, la lunghezza dell'interruzione può variare ampiamente. L'istruzione **SLEEP** di QuickBASIC offre ora un'alternativa migliore. (Consultare il capitolo 8, "Panoramica delle istruzioni e funzioni", per la sintassi di **SLEEP**.)

1.4.2 Cicli WHILE...WEND

Quando si conosce in anticipo il numero di esecuzioni di un ciclo, è vantaggioso utilizzare l'istruzione **FOR...NEXT**. Se ciò non è possibile, ma si conosce tuttavia la condizione che determinerà la chiusura del ciclo, allora è preferibile utilizzare l'istruzione **WHILE...WEND**. Invece di eseguire un conteggio per decidere se mantenere in esecuzione il ciclo, **WHILE...WEND** continua a ripetere il ciclo se la condizione del ciclo è vera.

La tabella 1.5 contiene la sintassi dell'istruzione **WHILE...WEND** e un esempio.

Tabella 1.5 Sintassi ed esempio di WHILE...WEND

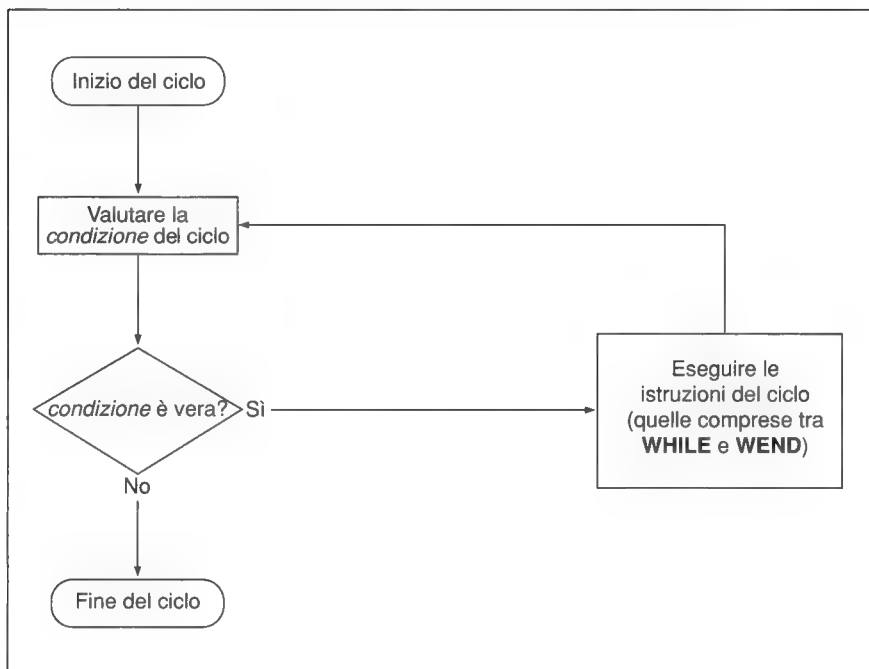
Sintassi	*Esempio*
WHILE *condizione*	`INPUT R$`
[*bloccoistruzioni*]	`WHILE R$ <> "S" AND R$ <> "N"`
WEND	`  PRINT "La risposta deve essere S o N."`
	`  INPUT R$`
	`WEND`

Esempio

L'esempio successivo assegna un valore iniziale di 10 alla variabile `X`, e poi continua a dimezzarlo finché il valore di `X` non è minore di 0,00001:

```
X = 10

WHILE X > .00001
   PRINT X
   X = X / 2
WEND
```

Nell'illustrazione 1.3 si vede il diagramma logico del ciclo **WHILE...WEND**.

Illustrazione 1.3 Diagramma logico del ciclo WHILE...WEND

1.4.3 Cicli DO...LOOP

Esattamente come l'istruzione **WHILE...WEND**, l'istruzione **DO...LOOP** esegue un blocco di istruzioni un numero indeterminato di volte; cioè, l'uscita dal ciclo dipende dalla veridicità o falsità della condizione del ciclo. Diversamente da **WHILE...WEND**, con **DO...LOOP** si può far terminare il ciclo sia con una condizione vera che con una falsa, e il test si può effettuare sia all'inizio che alla fine del ciclo.

La tabella 1.6 descrive la sintassi di un ciclo che effettua il test all'inizio.

Tabella 1.6 Sintassi ed esempio di DO...LOOP: test all'inizio del ciclo

Sintassi	*Esempio*

DO [{WHILE | UNTIL} *condizione*]
[*bloccoistruzioni-1*]
[EXIT DO
[*bloccoistruzioni-2*]]
LOOP
	`DO UNTIL INKEY$ <> ""`
	`' Un ciclo vuoto per`
	`' pausare finché non`
	`' viene premuto un tasto`
	`LOOP`

Le illustrazioni 1.4 e 1.5 mostrano i due tipi di istruzioni **DO...LOOP** che eseguono il test all'inizio del ciclo.

Illustrazione 1.4 Diagramma logico di DO WHILE...LOOP

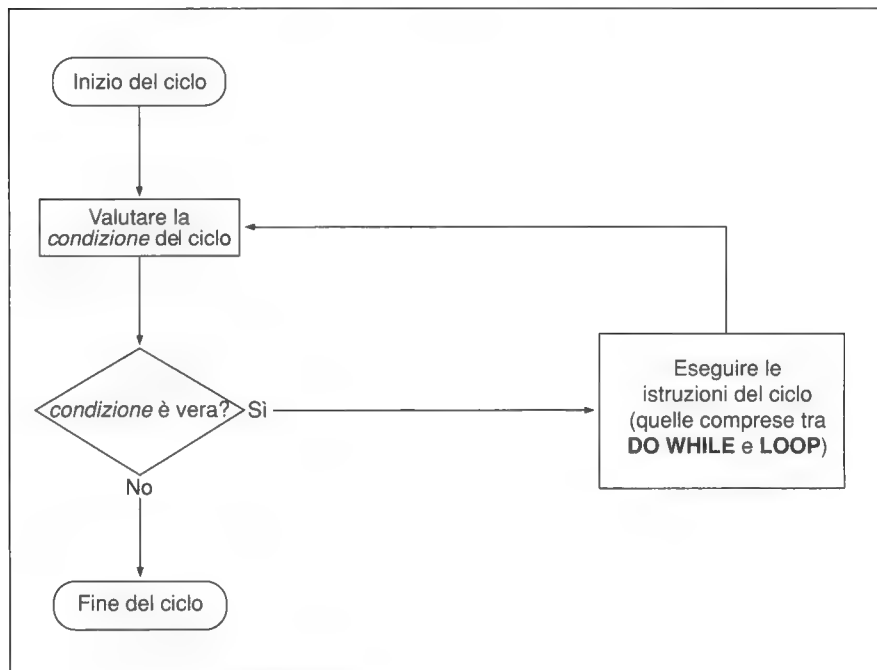



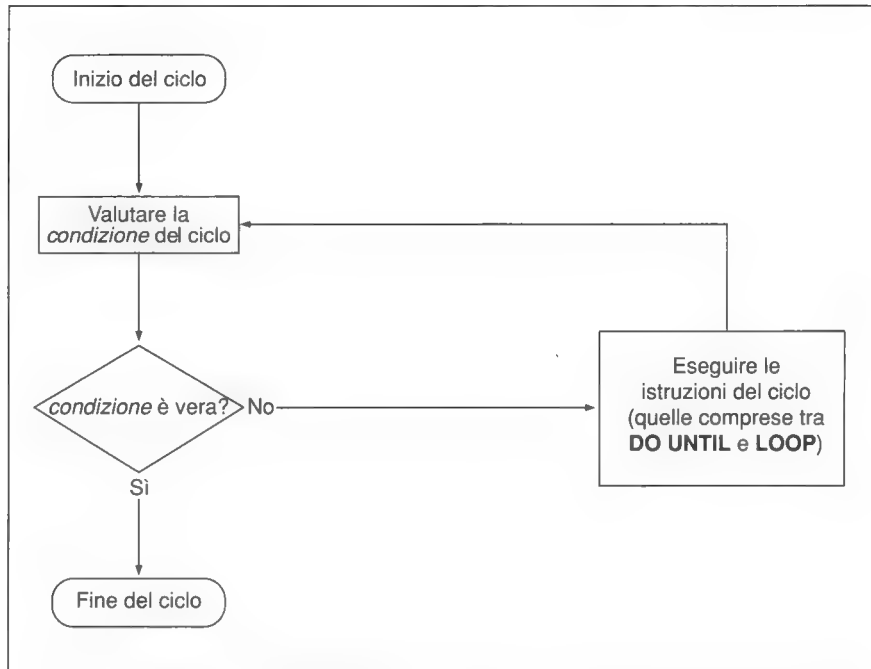
Illustrazione 1.5 Diagramma logico di DO UNTIL...LOOP

La tabella 1.7 illustra la sintassi di un ciclo che effettua il test alla fine del ciclo.

Tabella 1.7 Sintassi ed esempio di DO...LOOP: test alla fine del ciclo

Sintassi	***Esempio***
DO	`DO`
[*bloccoistruzioni-1*]	`  INPUT "Cambiamento:", Cam`
[EXIT DO	`  Totale = Totale + Cam`
[*bloccoistruzioni-2*]]	`LOOP WHILE Cam <> 0`
LOOP [{WHILE | UNTIL} *condizione*]	

Le illustrazioni 1.6 e 1.7 illustrano i due tipi di istruzioni che effettuano il test alla fine del ciclo.

Illustrazione 1.6 Diagramma logico di DO...LOOP WHILE

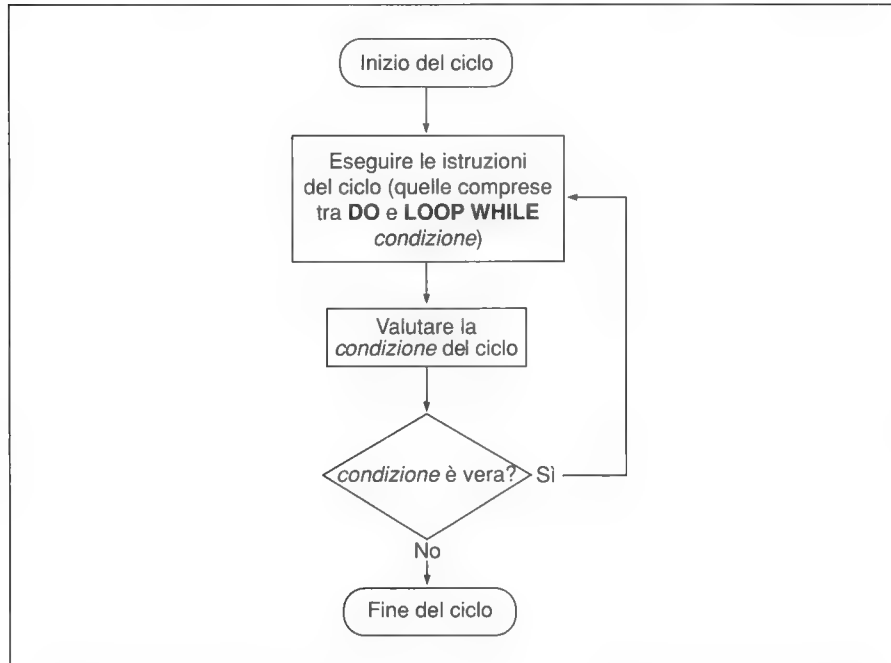



Illustrazione 1.7 Diagramma logico di DO...LOOP UNTIL

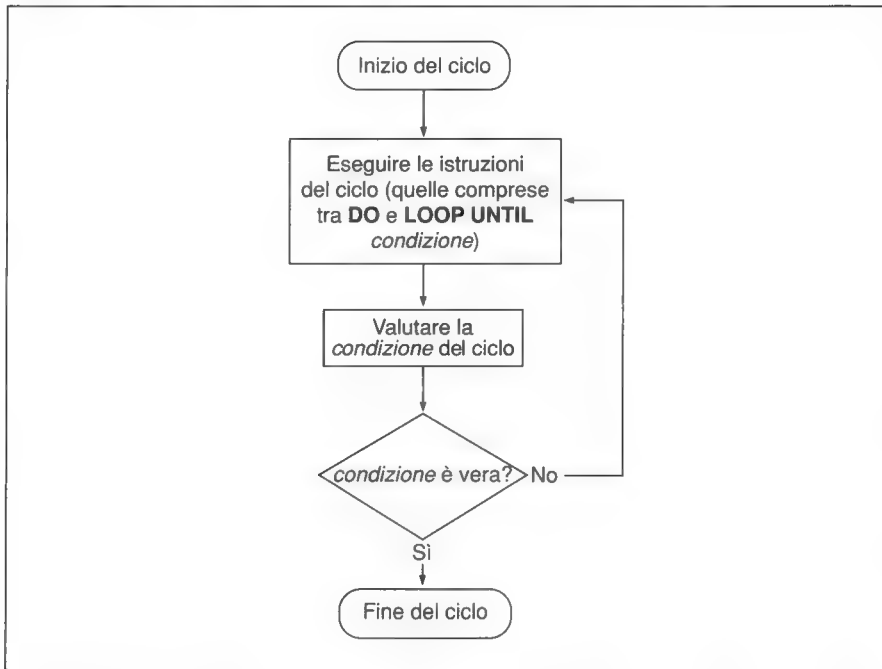
Inizio del ciclo

Eseguire le istruzioni del ciclo (quelle comprese tra **DO** e **LOOP UNTIL** *condizione*)

Valutare la *condizione* del ciclo

condizione è vera?

No

Sì

Fine del ciclo

1.4.3.1 Test di iterazione: come uscire da DO...LOOP

Porre il test del ciclo alla fine dell'istruzione **DO...LOOP** per creare un ciclo le cui istruzioni vengano comunque eseguite almeno una volta. Con l'istruzione **WHILE...WEND** talvolta si deve ricorrere all'espediente di preimpostare la variabile del ciclo ad un certo valore per garantire la prima esecuzione nel ciclc. Con **DO...LOOP**, tali espedienti non sono più necessari. Gli esempi successivi illustrano le due alternative.

```
' Il ciclo WHILE...WEND effettua il test all'inizio, per cui
' l'impostazione di Risposta$ a "S" è necessaria per forzare
' l'esecuzione del ciclo la prima volta:
Risposta$ = "S"
```

```
WHILE UCASE$(Risposta$) = "S"
   .
   .
   .
   INPUT "Eseguire nuovamente?"; Risposta$
WEND

' Lo stesso ciclo utilizzando DO...LOOP per effettuare il
' test alla fine del ciclo:
DO
   .
   .
   .
   INPUT "Eseguire nuovamente?"; Risposta$
LOOP WHILE UCASE$(Risposta$) = "S"
```

E' anche possibile riscrivere una condizione espressa con **WHILE** utilizzando **UNTIL**, come nell'esempio seguente:

```
'============================================================
'              Utilizzo di DO WHILE NOT...LOOP
'============================================================

' Se la fine del file 1 non è stata raggiunta, legge
' una riga dal file e la visualizza sullo schermo:
DO WHILE NOT EOF(1)
   LINE INPUT #1, BufferRiga$
   PRINT BufferRiga$
LOOP

'============================================================
'               Utilizzo di DO UNTIL...LOOP
'============================================================

' Finché non raggiunge la fine del file 1, legge
' una riga dal file e la visualizza sullo schermo:
DO UNTIL EOF(1)
   LINE INPUT #1, BufferRiga$
   PRINT BufferRiga$
LOOP
```

1.4.3.2 EXIT DO: un altro modo per uscire da DO...LOOP

All'interno di un'istruzione **DO...LOOP**, vengono eseguite altre istruzioni che possono modificare la condizione del test di iterazione da vera a falsa, o viceversa, e chiudere il ciclo. Negli esempi **DO...LOOP** presentati finora, il test viene eseguito o all'inizio o alla fine del ciclo. Utilizzando l'istruzione **EXIT DO** è però possibile effettuare il test per uscire dal ciclo in qualunque punto del ciclo. Una singola istruzione **DO...LOOP** può contenere un numero qualsiasi di istruzioni **EXIT DO** che possono apparire in qualsiasi punto del ciclo.

Esempio

L'esempio successivo apre un file e lo legge riga per riga fino alla fine, o finché non incontra una stringa specificata dall'utente. Se incontra la stringa prima di raggiungere la fine del file, un'istruzione **EXIT DO** fa uscire dal ciclo.

```
INPUT "File in cui cercare: ", File$
IF File$ = "" THEN END

INPUT "Stringa da cercare: " Stringa$
OPEN File$ FOR INPUT AS #1

DO UNTIL EOF(1)        ' EOF restituisce un valore di vero
                       ' se si è raggiunta la fine del file.
   LINE INPUT #1, RigaTemp$
   IF INSTR(RigaTemp$, Stringa$) > 0 THEN

      ' Visualizza la prima riga contenente la stringa ed
      ' esce dal ciclo:
      PRINT RigaTemp$
      EXIT DO
   END IF
LOOP
```

1.5 Programmi esempio

Gli esempi di applicazione forniti nel presente capitolo sono un programma bilancio conto corrente ed un programma che assicura che ciascuna riga di un file di testo termini con un ritorno a capo e un'interlinea.

1.5.1 Programma bilancio conto corrente (BILCC.BAS)

Questo programma chiede all'utente il bilancio iniziale del conto corrente e tutte le operazioni – depositi o versamenti – che sono avvenute. Successivamente il programma visualizza un elenco delle operazioni ed il bilancio finale del conto.

Istruzioni e funzioni utilizzate

Il programma dimostra le seguenti istruzioni trattate nel presente capitolo:

- **DO...LOOP WHILE**
- **FOR...NEXT**
- **EXIT FOR**
- Blocco **IF...THEN...ELSE**

Listato del programma

```
DIM Quantit(1 TO 100)
CONST FALSO = 0, VERO = NOT FALSO

' Ottiene il bilancio iniziale del conto:
CLS
INPUT "Digitare il bilancio iniziale e premere <INVIO>: _
   ", Bilancio

' Ottiene operazioni. Continua ad accettare input finché
' l'input non è zero per un'operazione, fino a un limite di
' 100 operazioni:
FOR NumOperaz% = 1 TO 100
   PRINT NumOperaz%;
   PRINT ") Digitare il valore dell'operazione ";
   PRINT " (oppure 0 per uscire): ";
   INPUT "", Quantit(NumOperaz%)
      IF Quantit(NumOperaz%) = 0 THEN
      NumOperaz% = NumOperaz% - 1
      EXIT FOR
   END IF
NEXT
```

```
' Ordina le operazioni in ordine crescente, usando
' un ordinamento per permutazioni:
Limite% = NumOperaz%
DO
   Scambi% = FALSO
   FOR I% = 1 TO (Limite% - 1)

      ' Se due elementi adiacenti non sono in ordine,
      ' permutarli:
      IF Quantit(I%) < Quantit(I% + 1) THEN
         SWAP Quantit(I%), Quantit(I% + 1)
         Scambi% = I%
      END IF
   NEXT I%

   ' Al passaggio successivo, non ordinare oltre il
   ' punto dell'ultima permutazione:
IF Scambi% THEN Limite% = Scambi%

' Ordina finché non avvengono più permutazioni:
LOOP WHILE Scambi%

' Stampa la matrice ordinata delle operazioni. Se una
' operazione è maggiore di zero, la rappresenta come
' "ATTIVO"; se è minore di zero, la rappresenta come
' "PASSIVO":
FOR I% = 1 TO NumOperaz%
   IF Quantit(I%) > 0 THEN
      PRINT USING "ATTIVO: $$#####.##"; Quantit(I%)
   ELSEIF Quantit(I%) < 0 THEN
      PRINT USING "PASSIVO: $$#####.##"; Quantit(I%)
   END IF

   ' Aggiorna il bilancio:
   Bilancio = Bilancio + Quantit(I%)
NEXT I%

' Stampa il bilancio finale:
PRINT
PRINT "---------------------------"
PRINT USING "Totale finale: $$######.##"; Bilancio
END
```

1.5.2 Filtro del ritorno a capo e dell'interlinea (CRLF.BAS)

Alcuni file di testo vengono salvati in un formato che utilizza solo un ritorno a capo (ritorno all'inizio della riga) o solo un'interlinea (passaggio alla riga successiva) per indicare la fine di una riga. Molti editor di testo sostituiscono il singolo ritorno a capo (CR) o interlinea (LF) con la sequenza dei due comandi (CR-LF), ogniqualvolta viene caricato un file da modificare. Se l'editor di testo non esegue la sostituzione di singoli CR e LF con la sequenza CR-LF, potrebbe essere necessario modificare il file in modo che alla fine di ogni riga ci sia la sequenza giusta.

Il programma seguente è un filtro che apre un file, sostituisce la combinazione CR-LF al posto di singoli CR o LF, e inserisce in un nuovo file le righe modificate. Il contenuto originale del file viene salvato in un file con estensione .BAK.

Istruzioni e funzioni utilizzate

Il programma dimostra le seguenti istruzioni trattate nel presente capitolo:

- **DO...LOOP WHILE**
- **DO UNTIL...LOOP**
- Blocco **IF...THEN...ELSE**
- **SELECT CASE...END SELECT**

Perché il programma sia di maggiore utilità, esso si avvale di alcune strutture non trattate nel presente capitolo:

- Una procedura **FUNCTION** chiamata `Backup$` che crea il file con estensione .BAK.

  Per ulteriori informazioni riguardanti la definizione e l'utilizzo delle procedure, consultare il capitolo 2, "Procedure SUB e FUNCTION".
- Una routine per la gestione degli errori chiamata `GestErr`, la quale provvede a correggere gli eventuali errori commessi dall'utente durante la digitazione del nome del file. Ad esempio, se l'utente digita il nome di un file inesistente, questa routine chiede all'utente un nome nuovo. Senza l'apporto di tale routine, un errore di questo genere chiuderebbe il programma.

  Per ulteriori informazioni relative alla gestione degli errori, consultare il capitolo 6, "Gestione degli errori e degli eventi".

Listato del programma

```
DEFINT A-Z  ' Il tipo predefinito per le variabili è intero.

' La funzione Backup$ crea un file backup con lo stesso
' nome di base di NomeFile$, ma con l'estensione .BAK:
DECLARE FUNCTION Backup$(NomeFile$)

' Inizializza le costanti e le variabili simboliche:
CONST FALSO = 0, VERO = NOT FALSO

Invio$ = CHR$(13)
Interlinea$ = CHR$(10)

DO
   CLS

   ' Ottiene il nome del file da cambiare:
   INPUT "Qual è il file da convertire"; FileOrig$

   ' Ottiene il nome del file backup:
   FileDest$ = Backup$(FileOrig$)

   ' Attiva la gestione di errori:
   ON ERROR GOTO GestErrori

   ' Copia il file di origine al file backup:
   NAME FileOrig$ AS FileDest$

   ' Disattiva la gestione di errori:
ON ERROR GOTO 0

' Apre il file backup per l'input e il file di
' origine per l'output:
OPEN FileDest$ FOR INPUT AS #1
OPEN FileOrig$ FOR OUTPUT AS #2

' La variabile PrecInvio è un flag che viene impostato su
' VERO quando il programma legge un carattere di INVIO:
PrecInvio = FALSO

' Legge dal file di origine finché non incontra la fine
' del file:
DO UNTIL EOF(1)
```

```
' Non è la fine del file, quindi legge un carattere:
CarFile$ = INPUT$(1, #1)

SELECT CASE CarFile$

   CASE Invio$            ' Il carattere è INVIO.

      ' Se anche il carattere precedente era INVIO,
      ' antepone INTERLINEA al carattere:
      IF PrecInvio THEN
         CarFile$ = Interlinea$ + CarFile$
      END IF

      ' In ogni caso, imposta la variabile PrecInvio a VERO:
      PrecInvio = VERO

   CASE Interlinea$       ' Il carattere è INTERLINEA.

      ' Se il carattere precedente non era INVIO,
      ' antepone INVIO al carattere:
      IF NOT PrecInvio THEN
         CarFile$ = Invio$ + CarFile$
      END IF

      ' In ogni caso, imposta la variabile PrecInvio
      ' a FALSO:
      PrecInvio = FALSO

   CASE ELSE              ' Né INVIO né INTERLINEA.

      ' Se il carattere precedente era INVIO, imposta
      ' la variabile PrecInvio a FALSO ed antepone
      ' INTERLINEA al carattere corrente:
      IF PrecInvio THEN
         PrecInvio = FALSO
         CarFile$ = Interlinea$ + CarFile$
      END IF

   END SELECT
      ' Registra i(l) caratteri(e) nel nuovo file:
      PRINT #2, CarFile$;
   LOOP
```

```
    ' Registra INTERLINEA se l'ultimo carattere del file
    ' è INVIO:
    IF PrecInvio THEN PRINT #2, Interlinea$;

    CLOSE                ' Chiude i due file.
    ' Chiede se si desidera proseguire:
    PRINT "Un altro file? (Sì/No)"
    ' Cambia l'input in maiuscolo:
    Ancora$ = UCASE$(INPUT$(1))
' Continua il programma se l'utente ha digitato "s" o "S":
LOOP WHILE Ancora$ = "S"
END

GestErrori:              ' Routine per la gestione di errori
CONST NESSUNFILE = 53, FILEESISTE = 58

    ' La funzione ERR restituisce il codice di errore
    ' dell'ultimo errore:
    SELECT CASE ERR
       CASE NESSUNFILE
       ' Il programma non ha trovato un file di origine
       ' con il nome specificato.
          PRINT "Impossibile trovare il file nella ";
          PRINT "directory corrente."
          INPUT "Digitare un nuovo nome: ", FileOrig$
          FileDest$ = Backup$(FileOrig$)
          RESUME
       CASE FILEESISTE
       ' Esiste già un file di nome <nomefile> .BAK
       ' nella directory: lo rimuove, e continua.
          KILL FileDest$
          RESUME

       CASE ELSE
       ' E' avvenuto un errore imprevisto: ferma
       ' il programma.
          ON ERROR GOTO 0
    END SELECT
```

```
' ======================== BACKUP$ ==========================
' Questa procedura restituisce un nome di file che consiste
' del nome di base del file di input (i caratteri prima
' del "."), più l'estensione ".BAK".
' ==========================================================

FUNCTION Backup$(NomeFile$) STATIC

   ' Cerca il punto fermo:
   Estensione = INSTR(NomeFile$, ".")

   ' Se trova il punto fermo, aggiunge .BAK alla base:
   IF Estensione > 0 THEN
   Backup$ = LEFT$(NomeFile$, Estensione - 1) + ".BAK"

   ' Se non lo trova, aggiunge .BAK all'intero nome:
   ELSE
      Backup$ = NomeFile$ + ".BAK"
   END IF
END FUNCTION
```

2 Procedure SUB e FUNCTION

Il presente capitolo spiega come semplificare la programmazione mediante la scomposizione dei programmi in componenti logiche più piccole. Queste componenti – conosciute con il nome di "procedure" – possono diventare in seguito unità modulari che potenziano ed ampliano lo stesso linguaggio BASIC.

Alla fine di questo capitolo, l'utente sarà in grado di:

- Definire e chiamare procedure in BASIC
- Utilizzare variabili locali e globali nelle procedure
- Utilizzare le procedure al posto delle subroutine **GOSUB** e delle funzioni **DEF FN**
- Passare argomenti alle procedure e leggerne i valori restituiti
- Scrivere procedure ricorsive (procedure che chiamano se stesse)

Nonostante si possa creare un programma con qualsiasi editor di testo, l'editor di QuickBASIC semplifica notevolmente la stesura di programmi contenenti procedure. Inoltre (durante la memorizzazione del programma), QuickBASIC generalmente produce automaticamente un'istruzione **DECLARE**. Quest'istruzione garantisce il passaggio del numero e del tipo esatto di argomenti ad una procedura e permette al programma di chiamare procedure che sono state definite in altri moduli.

2.1 Procedure: unità modulari per la programmazione

Il termine "procedura" viene utilizzato in questo capitolo per le costruzioni **SUB...END SUB** e **FUNCTION...END FUNCTION**. Conviene utilizzare le procedure per raggruppare processi reiterati. Per esempio, nella stesura di un programma di applicazione autonomo capace di leggere argomenti dalla riga di comando, sarà utile assegnare a una procedura il compito di scomporre in due o più argomenti la stringa restituita dalla funzione **COMMAND$**. Una volta messa a punto, questa procedura si potrà utilizzare anche in altri programmi. In altri termini, la creazione di procedure consente al BASIC di rispondere sempre di più alle esigenze dell'utente.

La programmazione con procedure presenta due notevoli vantaggi:

- Le procedure permettono di scomporre il programma in unità logiche, ciascuna delle quali può essere messa a punto più facilmente rispetto ad un intero programma senza procedure.
- Le procedure di un programma possono essere utilizzate, spesso senza alcuna modifica, come unità modulari in altri programmi.

Le procedure si possono inserire anche in una libreria Quick, un file speciale che si può caricare in memoria quando si avvia QuickBASIC. Una volta memorizzato il contenuto di una libreria, qualsiasi programma può accedere alle procedure della libreria. Ciò facilita sia la condivisione che il salvataggio dei programmi. (Per ulteriori informazioni relative alla creazione delle librerie Quick, consultare l'appendice H.)

2.2 Confronto di procedure con subroutine e funzioni

Se si ha familiarità con le precedenti versioni di QuickBASIC, una procedura **SUB...END SUB** potrebbe apparire simile a una subroutine **GOSUB...RETURN**. Si noterà anche una certa similitudine tra una procedura **FUNCTION...END FUNCTION** ed una funzione **DEF FN...END DEF**. Le procedure hanno comunque molti vantaggi rispetto a queste costruzioni precedenti, come spiegato nelle sezioni 2.2.1 e 2.2.2.

Nota Per evitare di fare confusione tra una procedura **SUB** e la destinazione di un'istruzione **GOSUB**, nel presente manuale si fa riferimento alle procedure **SUB** con il nome "sottoprogrammi", e ai blocchi d'istruzioni, cui si accede mediante l'istruzione **GOSUB...RETURN**, con il nome "subroutine".

2.2.1 Confronto tra SUB e GOSUB

Sebbene l'utilizzazione delle subroutine **GOSUB** agevoli la scomposizione dei programmi in unità facilmente gestibili, le procedure **SUB** hanno numerosi vantaggi rispetto alle subroutine, come è discusso in seguito.

2.2.1.1 Variabili locali e globali

Nelle procedure **SUB**, tutte le variabili sono locali come predefinizione; cioè esse hanno un'area di validità che coincide con quella della definizione della procedura **SUB**. Per esempio, la variabile `I` nel sottoprogramma `Test` dell'esempio seguente è una variabile locale di `Test`, e non è collegata con la variabile `I` nel codice a livello di modulo:

```
I = 1
CALL Test
PRINT I         ' I è ancora uguale a 1.
END

SUB Test STATIC
   I = 50
END SUB
```

Una subroutine **GOSUB** ha un grosso svantaggio se utilizzata nei programmi come un'unità modulare, e cioè adopera solo "variabili globali". Con le variabili globali, due variabili `I` nello stesso modulo, sono sempre uguali anche se una è interna e l'altra esterna ad una subroutine. Ogni modifica del valore di una variabile `I` interna alla subroutine, cambia tutte le `I` in qualsiasi parte del modulo. Ne risulta che, per spostare una subroutine da un modulo ad un'altro, è spesso necessario assegnare alle variabili della subroutine un nome nuovo per evitare conflitti con i nomi delle variabili del nuovo modulo.

2.2.1.2 Utilizzo nei programmi a più moduli

Una procedura **SUB** può essere definita in un modulo e chiamata da un altro. Questo riduce in modo significativo la quantità di codice richiesto per un programma e rende più facile la condivisione di codice tra numerosi programmi.

Una subroutine **GOSUB** deve invece essere definita ed utilizzata nello stesso modulo.

2.2.1.3 Operazioni su diversi insiemi di variabili

Una procedura **SUB** può essere chiamata all'interno di un programma numerose volte, per ognuna delle quali le viene passato un diverso insieme di variabili. Ciò si ottiene chiamando la procedura **SUB** con un elenco di argomenti. (Per ulteriori informazioni sulla modalità di quest'operazione, consultare la sezione 2.5, "Passaggio di argomenti a procedure".) Nell'esempio successivo, il sottoprogramma `Confronta` viene chiamato due volte, in ognuna delle quali gli viene passato un diverso insieme di variabili:

```
X = 4 : Y = 5

CALL Confronta (X, Y)

Z = 7 : W = 2
CALL Confronta (Z, W)
END

SUB Confronta (A, B)
   IF A < B THEN SWAP A, B
END SUB
```

E' invece difficile chiamare una subroutine **GOSUB** più volte all'interno di uno stesso programma, con differenti insiemi di variabili. Il procedimento richiede di copiare i valori a e da variabili globali, come si vede dal prossimo esempio:

```
X = 4 : Y = 5
A = X : B = Y
GOSUB Confronta
X = A : Y = B

Z = 7 : W = 2
A = Z : B = W
GOSUB Confronta
Z = A : W = B
END

Confronta:
   If A < B THEN Swap A, B
RETURN
```

2.2.2 Confronto tra FUNCTION e DEF FN

Mentre la definizione multilinea della funzione **DEF FN** risponde alle esigenze di funzioni più complesse di quelle che possono trovare spazio in una monoriga, le procedure **FUNCTION**, oltre a questo vantaggio, ne posseggono altri, descritti qui sotto.

2.2.2.1 Variabili locali e globali

Tutte le variabili interne ad una procedura **FUNCTION** sono locali ad essa per definizione, sebbene esista la possibilità di utilizzare anche variabili globali. (Per ulteriori informazioni relative alle procedure ed alle variabili globali, consultare la sezione 2.6, "Variabili condivise con SHARED".)

Per definizione, le variabili utilizzate all'interno della funzione **DEF FN** sono globali rispetto al modulo corrente (ciò vale anche per le subroutine **GOSUB**). Si può comunque rendere locale una variabile interna ad una funzione **DEF FN**, citandola in un'istruzione **STATIC**.

2.2.2.2 Come cambiare le variabili passate alle procedure

Il passaggio delle variabili alle procedure **FUNCTION** viene effettuato per riferimento o per valore. Quando il passaggio avviene per riferimento, è possibile modificare la variabile cambiando il relativo parametro all'interno della procedura stessa. Ad esempio, nel programma seguente, dopo aver chiamato `OttResto`, il valore di `X` diventa 2, dal momento che 2 è il valore di `M` alla fine della **FUNCTION**:

```
X = 89
Y = 40
PRINT OttResto (X, Y)
PRINT X, Y          ' Ora X è 2.
END

FUNCTION OttResto (M, N)
   OttResto = M MOD N
   M = M \ N
END FUNCTION
```

Il passaggio delle variabili alla funzione **DEF FN** avviene solo per valore, pertanto `FNResto` modifica `M` senza incidere su `X`:

```
DEF FNResto (M, N)
   FNResto = M MOD N
   M = M \ N
END DEF

X = 89
Y = 40
PRINT FNResto (X, Y)

PRINT X, Y          ' X è ancora 89.
```

Per ulteriori informazioni relative alla differenza tra un passaggio effettuato per riferimento ed un passaggio effettuato per valore, consultare le sezioni 2.5.5 e 2.5.6.

2.2.2.3 Chiamata di una procedura dall'interno della sua definizione

Una procedura **FUNCTION** può essere "ricorsiva" nel senso che può chiamarsi dall'interno della propria definizione. (Per ulteriori informazioni sul modo in cui le procedure possono essere ricorsive, consultare la sezione 2.9.) Al contrario, una funzione **DEF FN** non può essere ricorsiva.

2.2.2.4 Utilizzo nei programmi a più moduli

E' possibile definire una procedura in un modulo e poi utilizzarla in un altro modulo, nel quale sia stata precedentemente citata in un'istruzione **DECLARE**. Se non si effettua questa operazione, il programma riterrà che il nome della **FUNCTION** si riferisca ad una variabile. (Per ulteriori informazioni sull'utilizzazione di **DECLARE** in questo senso, consultare la sezione 2.5.4, "Verifica degli argomenti con l'istruzione DECLARE".)

Una funzione **DEF FN** può però essere utilizzata solo nel modulo nel quale è stata definita. Inoltre, mentre le procedure **SUB** e **FUNCTION** possono essere chiamate prima di comparire nel programma, una funzione **DEF FN** deve essere sempre definita prima di essere utilizzata in un modulo.

Nota Per denominare una procedura **FUNCTION** si può utilizzare qualsiasi nome valido di variabile BASIC, tranne quelli con iniziale **FN**. Il nome di una funzione **DEF FN** deve essere sempre preceduto da **FN**.

2.3 Definizione di procedure

Le definizioni di una procedura BASIC hanno la seguente sintassi generale:

{SUB | FUNCTION} *nomeprocedura* [(*elencoparametri*)] **[STATIC]**
 [*bloccoistruzioni-1*]
 [EXIT {SUB | FUNCTION}
 [*bloccoistruzioni-2*]]
END {SUB | FUNCTION}

L'elenco seguente descrive le parti della definizione di una procedura:

Parte	*Descrizione*
{SUB | FUNCTION}	Indica rispettivamente l'inizio di una procedura **SUB** oppure **FUNCTION**.
nomeprocedura	Qualsiasi nome di variabile valido, lungo fino a 40 caratteri. Lo stesso nome non può essere utilizzato per una **SUB** e per una **FUNCTION**.
elencoparametri	Un elenco di variabili, separate da virgole, che mostra il numero ed il tipo degli argomenti da passare alle procedure. (La sezione 2.5.1 spiega la differenza tra parametri ed argomenti.)
STATIC	Utilizzando l'attributo **STATIC**, le variabili locali sono considerate variabili **STATIC**; ciò significa che conservano i loro valori tra una chiamata della procedura e l'altra. Omettendo l'attributo **STATIC**, le variabili locali sono "automatiche" per predefinizione; cioè sono inizializzate allo zero o alla stringa nulla all'inizio di ogni chiamata della procedura. Per ulteriori informazioni, consultare la sezione 2.7, "Variabili automatiche e variabili STATIC".
END {SUB | FUNCTION}	Termina la definizione di una **SUB** o di una **FUNCTION**. Per essere eseguita in maniera corretta, ogni procedura deve avere esattamente una istruzione **END {SUB | FUNCTION}**. Quando il programma incontra un'istruzione **END SUB** o **END FUNCTION**, esce dalla procedura e ritorna all'istruzione immediatamente successiva a quella che aveva chiamato la procedura. Per uscire dalla procedura si possono anche utilizzare una o più istruzioni opzionali **EXIT {SUB | FUNCTION}**.

All'interno della definizione di una procedura, sono permesse tutte le espressioni e le istruzioni valide del BASIC, ad eccezione delle seguenti:

- **DEF FN...END DEF**, **FUNCTION...END FUNCTION**, e **SUB...END SUB**

  Non è possibile nidificare le definizioni di procedura o definire una funzione **DEF FN** dentro un procedura. Tuttavia, una procedura può chiamare un'altra procedura o una funzione **DEF FN**.
- **COMMON**
- **DECLARE**
- **DIM SHARED**
- **OPTION BASE**
- **TYPE...END TYPE**

Esempio

L'esempio seguente mostra una procedura **FUNCTION** denominata `PotenzaIntera`:

```
FUNCTION PotenzaIntera& (X&, Y&) STATIC
   ValPotenza& = 1
   FOR I& = 1 TO Y&
      ValPotenza& = ValPotenza& * X&
   NEXT I&
   PotenzaIntera& = ValPotenza&
END FUNCTION
```

2.4 Chiamata di procedure

La differenza tra la chiamata di una procedura **FUNCTION** e di una procedura **SUB** viene spiegata nelle due sezioni seguenti.

2.4.1 Chiamata di una procedura FUNCTION

Una procedura **FUNCTION** viene chiamata allo stesso modo di una funzione intrinseca del BASIC come **ABS**, cioè usandone il nome all'interno di un'espressione, come si vede qui:

```
' Tutte le istruzioni seguenti chiamano una FUNCTION
' denominata "ToDec":
PRINT 10 * ToDec
X = ToDec
IF ToDec = 10 THEN PRINT "Esterno all'intervallo."
```

Una **FUNCTION** può restituire valori modificando le variabili ad essa passate come argomenti. (Per una spiegazione sulle modalità dell'operazione, consultare la sezione 2.5.5, "Passaggio di argomenti per riferimento".) Inoltre, una **FUNCTION** restituisce un singolo valore assegnato al nome stesso della **FUNCTION**, che deve quindi corrispondere al tipo dati del valore restituito. Per esempio, una **FUNCTION** che restituisce valori a stringa deve avere il simbolo (**$**) alla fine del nome, oppure è necessario dichiararne il tipo a stringa con un'istruzione **DEFSTR**.

Esempio

Il programma seguente presenta una procedura **FUNCTION** che restituisce un valore a stringa. Notare che il suffisso utilizzato come simbolo del tipo a stringa (**$**) è parte integrante del nome della procedura.

```
Motto$ = OttInput$            ' Chiama la FUNCTION e ne
                              ' assegna il valore restituito
                              ' ad una variabile a stringa.
PRINT Motto$                  ' Stampa la stringa.
END

' ======================== OttInput$ ========================
'  Il carattere $ di dichiarazione di tipo alla fine del
'  nome di questa FUNCTION significa che restituisce un
'  valore a stringa.
' ==========================================================

FUNCTION OttInput$ STATIC

   ' Restituisce una stringa di 10 caratteri letta dalla
   ' tastiera, e rivisualizza sullo schermo ciascun
   ' carattere digitato:
   FOR I% = 1 TO 10
      Car$ = INPUT$(1)        ' Legge il carattere.
      PRINT Car$;             ' Rivisualizza il carattere
                              ' sullo schermo.
      Temp$ = Temp$ + Car$    ' Aggiunge il carattere alla
                              ' stringa.
   NEXT
   PRINT
   OttInput$ = Temp$          ' Assegna la stringa alla
                              ' FUNCTION.
END FUNCTION
```

Nota L'esempio di programma sopra illustrato può essere utilizzato solo in ambiente QuickBASIC, e non può essere compilato con il comando BC da DOS.

2.4.2 Chiamata di una procedura SUB

Una procedura **SUB** si differenzia da una procedura **FUNCTION** in quanto una **SUB** non può essere chiamata semplicemente citandone il nome all'interno di un'espressione. La chiamata di una procedura **SUB** è un'istruzione a sé, come l'istruzione **CIRCLE** del BASIC. Inoltre, una **SUB** non restituisce un valore assegnato al suo nome come fa invece una **FUNCTION**. Una **SUB** tuttavia può modificare il valore di qualsiasi variabile ad essa passata, allo stesso modo di una **FUNCTION**. La sezione 2.5.5, "Passaggio di argomenti per riferimento", spiega le modalità di quest'operazione.

Una procedura **SUB** può essere chiamata in due diversi modi:

- Inserendone il nome in un'istruzione **CALL**:

  ```
  CALL StampaMessaggio
  ```

- Utilizzandone il nome come un'istruzione a se stante:

  ```
  StampaMessaggio
  ```

Se viene omessa la parola chiave **CALL**, non racchiudere tra parentesi gli argomenti passati alla procedura **SUB**:

```
' Chiama il sottoprogramma ElabInput con CALL passando i
' tre argomenti Primo$, Secondo$, e NumArg%:
CALL ElabInput (Primo$, Secondo$, NumArg%)

' Chiama il sottoprogramma ElabInput senza CALL passando
' gli stessi argomenti (notare che l'elenco di argomenti
' non è racchiuso tra parentesi):
ElabInput Primo$, Secondo$, NumArg%
```

Per ulteriori informazioni sul passaggio degli argomenti alle procedure, consultare la sezione 2.5.

Se si possiede un programma che non utilizza **CALL** per chiamare le procedure **SUB**, e se non si utilizza QuickBASIC per la stesura del programma, è necessario inserire il nome di ogni procedura **SUB** in un'istruzione **DECLARE** prima che essa venga chiamata:

```
DECLARE SUB RilevaTasto
.
.
.
RilevaTasto
```

Questo risulta necessario comunque solo se si sviluppano programmi fuori dell'ambiente QuickBASIC, dal momento che QuickBASIC inserisce automaticamente, quando salva un programma, istruzioni **DECLARE** ovunque esse siano necessarie.

2.5 Passaggio di argomenti a procedure

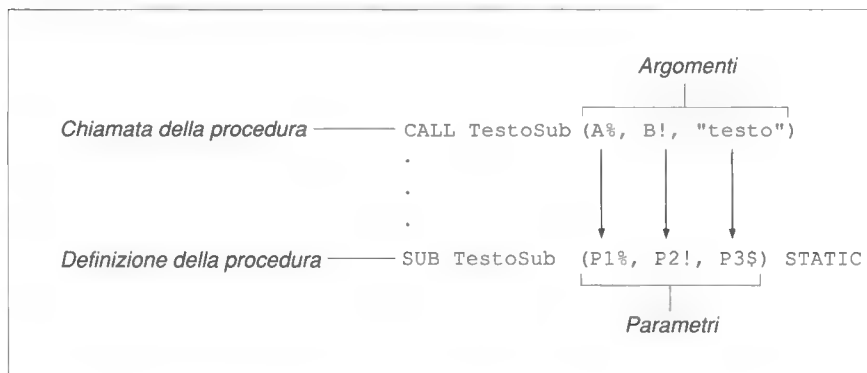
Le sezioni da 2.5.1 a 2.5.4 spiegano come distinguere i parametri dagli argomenti, come effettuare il passaggio degli argomenti alle procedure, e come verificare che gli argomenti siano del tipo e del numero giusti.

2.5.1 Parametri ed argomenti

Prima di poter effettuare il passaggio di argomenti alle procedure, è necessario aver chiara la differenza tra i termini "parametro" ed "argomento":

Parametro	*Argomento*
Il nome di una variabile che compare in un'istruzione **SUB**, **FUNCTION** o **DECLARE**	Una costante, una variabile o un'espressione passata ad una **SUB** o **FUNCTION** quando questa viene chiamata

Nella definizione di una procedura, i parametri prendono il posto degli argomenti che verranno sostituiti durante l'esecuzione. Come mostra l'illustrazione 2.1, ogni volta che la procedura viene chiamata, il primo parametro riceve il valore del primo argomento, il secondo parametro riceve il valore del secondo argomento, e così via.

Illustrazione 2.1 Parametri ed argomenti

L'illustrazione 2.1 mostra anche un'altra importante regola: sebbene i nomi delle variabili in un elenco di argomenti possano differire da quelli nel corrispondente elenco di parametri, il numero dei parametri deve essere uguale a quello degli argomenti. Inoltre, gli argomenti ed i parametri corrispondenti devono essere dello stesso tipo (stringa, numero intero, numero in precisione semplice, ecc). Per ulteriori informazioni utili ad assicurare la corrispondenza del numero e dei tipi dei parametri e argomenti, consultare la sezione 2.5.4, "Verifica degli argomenti con l'istruzione DECLARE".

Qualunque delle voci seguenti, delimitata da virgole, può apparire in un elenco di parametri:

- Nomi validi di variabili, eccetto le stringhe a lunghezza fissa

  Ad esempio, `X$` e `X AS STRING` sono entrambe legittime in un elenco di parametri, poiché si riferiscono a stringhe a lunghezza variabile. Tuttavia `X AS STRING * 10` è una stringhe a lunghezza fissa di 10 caratteri e quindi non può comparire nell'elenco di parametri. (Le stringhe a lunghezza fissa possono essere tranquillamente passate come *argomenti* alle procedure. Per ulteriori informazioni sulle stringhe a lunghezza fissa e a lunghezza variabile, consultare il capitolo 4, "Manipolazione di stringhe".)
- Nomi di matrici seguite dalle due parentesi

Qualunque delle voci seguenti, delimitata da virgole, può apparire in un elenco di argomenti:

- Costanti
- Espressioni
- Nomi validi di variabili
- Nomi di matrici seguite dalle due parentesi

Esempi

L'esempio seguente mostra la prima riga della definizione di un sottoprogramma con il relativo elenco dei parametri:

```
SUB SubTest (A%, Matrice(), VarRec AS TipoRec, Cs$)
```

Il primo parametro, `A%`, è un intero; il secondo, `Matrice()`, è una matrice in precisione semplice, dal momento che le variabili numeriche sono per predefinizione in precisione semplice; il terzo parametro, `VarRec`, è un record di tipo `TipoRec`; il quarto parametro, `Cs$`, è una stringa.

La riga `CALL SubTest` nell'esempio successivo chiama il sottoprogramma `SubTest` e passa ad esso quattro argomenti del tipo adatto:

```
TYPE TipoRec
   Grado AS STRING * 12
   NumMatr AS LONG
END TYPE

DIM VarRec AS TipoRec

CALL SubTest (X%, A(), VarRec, "Daphne")
```

2.5.2 Passaggio di costanti ed espressioni

Le costanti – siano esse a stringa o numeriche – possono comparire nell'elenco degli argomenti passati alle procedure. Naturalmente, una costante a stringa deve essere passata ad un parametro a stringa ed una costante numerica ad un parametro numerico, come nel prossimo esempio:

```
CONST LargSchermo = 80
CALL StampaTitolo (LargSchermo, "Prospetto mensile")
.
.
.
SUB StampaTitolo (LS%, Titolo$)
   .
   .
   .
END SUB
```

Se una costante numerica in un elenco di argomenti non ha lo stesso tipo del corrispondente parametro nell'istruzione **SUB** o **FUNCTION**, la corrispondenza del tipo viene forzata, come si vede dall'output dell'esempio successivo:

```
CALL Test (4.6, 4.1)
END

SUB Test (x%, y%)
   PRINT x%, y%
END SUB
```

Output

```
5     4
```

Anche intere espressioni formate da operazioni su variabili e costanti possono essere passate ad una procedura. Come nel caso delle costanti, anche con le espressioni numeriche che non sono dello stesso tipo dei rispettivi parametri viene forzata la corrispondenza del tipo, come si vede qui:

```
Prova A! + 25!, NOT ValBooleano%

' Nella chiamata successiva, la variabile ad intero
' lungo ValB& una volta racchiusa tra parentesi diventa
' un'espressione. L'espressione (ValB&) è forzata in un
' intero breve all'interno della SUB Prova:
```

```
Prova A! / 3.1, (ValB&)
.
.
.
END

SUB Prova (Param1!, Param2%)
    .
    .
    .
END SUB
```

2.5.3 Passaggio di variabili

In questa sezione vengono trattate le modalità di passaggio alle procedure di variabili semplici, di intere matrici o record, e di singoli elementi di una matrice o di un record.

2.5.3.1 Passaggio di variabili semplici

Sia nell'elenco degli argomenti che in quello dei parametri, è possibile dichiarare il tipo di una variabile semplice in uno dei seguenti modi:

- Far seguire il nome della variabile da uno dei seguenti suffissi: **%**, **&**, **!**, **#**, o **$**.
- Dichiarare la variabile in una clausola *dichiara nomevariabile* **AS** *tipo*, dove *dichiara* può essere **DIM**, **COMMON**, **REDIM**, **SHARED** o **STATIC**, mentre il *tipo* può essere **INTEGER**, **LONG**, **SINGLE**, **DOUBLE**, **STRING** o **STRING** * *n*. Un esempio:

  ```
  DIM A AS LONG
  ```
- Utilizzare un'istruzione **DEF***tipo* per impostare il tipo predefinito.

A prescindere dal metodo scelto, le variabili corrispondenti devono avere lo stesso tipo nell'elenco degli argomenti e in quello dei parametri, come è mostrato nell'esempio successivo.

Esempio

In quest'esempio, vengono passati due argomenti alla procedura **FUNCTION**. Il primo è un intero che fornisce la lunghezza della stringa restituita dalla **FUNCTION**, mentre il secondo è un carattere che viene ripetuto per creare la stringa.

```
FUNCTION CarStringa$ (A AS INTEGER, B$) STATIC
   CarStringa$ = STRING$ (A%, B$)
END FUNCTION

DIM X AS INTEGER
INPUT "Digitare un numero (da 1 a 80): ", X
INPUT "Digitare un carattere: ", Y$

' Visualizza una stringa consistente nel carattere Y$,
' ripetuto un numero X di volte:
PRINT CarStringa$ (X, Y$)
END
```

Output

```
Digitare un numero (da 1 a 80): 21
Digitare un carattere: #
#####################
```

2.5.3.2 Passaggio di una matrice intera

Per passare tutti gli elementi di una matrice ad una procedura, inserire il nome della matrice seguito dalle parentesi, sia nell'elenco dei parametri che in quello degli argomenti.

Esempio

Quest'esempio mostra il passaggio di tutti gli elementi di una matrice ad una procedura:

```
DIM Valori(1 TO 5) AS INTEGER

' Notare le parentesi vuote dopo il nome della matrice nella
' chiamata della procedura ed il passaggio della matrice:
CALL CambiaMatrice (1, 5, Valori())
CALL StampaMatrice (1, 5, Valori())
END

' Notare le parentesi vuote dopo il parametro P:
SUB CambiaMatrice (Min%, Mass%, P() AS INTEGER) STATIC
   FOR I% = Min% TO Mass%
      P(I%) = I% ^ 3
   NEXT I%
END SUB
```

```
SUB StampaMatrice (Min%, Mass%, P() AS INTEGER) STATIC
   FOR I% = Min% TO Mass%
      PRINT P(I%)
   NEXT I%
   PRINT
END SUB
```

2.5.3.3 Passaggio di singoli elementi di una matrice

Se una procedura non richiede una matrice intera, è possibile passarle singoli elementi di questa. Per passare un elemento di una matrice, si utilizza il nome della matrice, seguito dal relativo indice racchiuso in parentesi.

Esempio

Nell'esempio seguente l'istruzione `RadQVal Matrice(4, 2)` passa al sottoprogramma `RadQVal` l'elemento della quarta riga e seconda colonna della matrice (notare come il sottoprogramma modifichi davvero il valore di questo elemento):

```
DIM Matrice(1 TO 5, 1 TO 3)

Matrice(4, 2) = -36
PRINT Matrice(4, 2)
RadQVal Matrice(4, 2)
PRINT Matrice(4, 2)           ' La chiamata di RadQVal ha
                              ' modificato il valore di
                              ' Matrice(4, 2).
END

SUB RadQVal(A) STATIC
   A = SQR (ABS(A))
END SUB
```

Output

```
-36
 6
```

2.5.3.4 Utilizzo delle funzioni sui limiti delle matrici

Le funzioni **LBOUND** ed **UBOUND** vengono utilizzate per determinare le dimensioni di una matrice passata ad una procedura. La funzione **LBOUND** trova il valore più piccolo dell'indice di una matrice, mentre la funzione **UBOUND** trova quello più grande. Queste funzioni evitano all'utente di dover passare ad una procedura i limiti di ciascuna dimensione della matrice.

Esempio

Nell'esempio seguente, il sottoprogramma utilizza la funzione **LBOUND** per inizializzare le variabili `Riga` e `Col` ai valori minimi degli indici in ciascuna dimensione di `A`, e la funzione **UBOUND** per limitare il numero di esecuzioni del ciclo **FOR** in base al numero degli elementi della matrice.

```
SUB Stampa(A(2)) STATIC
   FOR Riga = LBOUND(A,1) TO UBOUND(A,1)
      FOR Col = LBOUND(A,2) TO UBOUND(A,2)
         PRINT A(Riga,Col)
      NEXT Col
   NEXT Riga
END SUB
```

2.5.3.5 Passaggio di un intero record

Per passare un record intero (una variabile di un tipo definito dall'utente) ad una procedura, seguire le seguenti fasi:

1. Definire il tipo (`UnitaMagazz` in questo esempio):

   ```
   TYPE UnitaMagazz
      NumeroPezzo AS STRING * 6
      Descrizione AS STRING * 20
      PrezzoUnit AS SINGLE
      Quantita AS INTEGER
   END TYPE
   ```

2. Dichiarare una variabile (`RecordMagazz`) di questo tipo:

   ```
   DIM RecordMagazz AS UnitaMagazz
   ```

3. Chiamare una procedura (`TrovaRecord`) e passarvi la variabile che si è dichiarata:

   ```
   CALL TrovaRecord (RecordMagazz)
   ```

continua

4. Nella definizione della procedura, attribuire al parametro lo stesso tipo della variabile:

```
SUB TrovaRecord (VarRecord AS UnitaMagazz) STATIC
    .
    .
    .
END SUB
```

2.5.3.6 Passaggio di singoli elementi di un record

Se si vuole passare ad una procedura un singolo elemento di un record, si deve inserire il nome dell'elemento (*nomerecord.nomeelemento*) nell'elenco degli argomenti. Assicurarsi, come sempre, che il parametro corrispondente nella definizione della procedura si accordi per tipo a quell'elemento.

Esempio

L'esempio seguente mostra come passare alla procedura **SUB** `StampaEtich` i due elementi della variabile a record `UnitaMagazz`. Notare come ciascun parametro della procedura **SUB** si accordi con il tipo dei singoli elementi del record.

```
TYPE TipoMagazz
   NumeroPezzo AS STRING * 6
   Descriz AS STRING * 20
   PrezzoUnit AS SINGLE
   Quantita AS INTEGER
END TYPE

DIM UnitaMagazz AS TipoMagazz

CALL StampaEtich (UnitaMagazz.Descriz, _
   UnitaMagazz.PrezzoUnit)
.
.
.
END

SUB StampaEtich (Desc$, Prezzo AS SINGLE)
   .
   .
   .
END SUB
```

2.5.4 Verifica degli argomenti con l'istruzione DECLARE

Se si sta utilizzando QuickBASIC per la stesura di un programma, si noterà che QuickBASIC inserisce automaticamente un'istruzione **DECLARE** per ciascuna procedura ogni volta che si memorizza il programma. Ciascuna istruzione **DECLARE** consiste nella parola **DECLARE** seguita dalle parole **SUB** o **FUNCTION**, dal nome della procedura e da parentesi. Se la procedura non ha parametri, le parentesi sono vuote. In caso contrario, le parentesi racchiudono un *elencoparametri* che specifica il numero ed il tipo degli argomenti che devono essere passati alla procedura. Questo *elencoparametri* ha lo stesso formato dell'elenco nella riga di definizione della **SUB** o **FUNCTION**.

L'*elencoparametri* in un'istruzione **DECLARE** attiva il "controllo del tipo" degli argomenti passati alla procedura. Cioè, ogni volta che la procedura viene chiamata con variabili per argomento, verrà controllato che le variabili corrispondano per numero e tipo ai parametri nell'istruzione **DECLARE**.

Durante la memorizzazione di un programma, QuickBASIC inserisce tutte le definizioni di procedura alla fine del modulo. Di conseguenza, se non ci fossero istruzioni **DECLARE** e si compilasse questo programma con il comando BC, si verificherebbe il problema del "riferimento anticipato" (chiamare una procedura prima che sia definita). Producendo un prototipo della definizione della procedura, le istruzioni **DECLARE** permettono al programma di chiamare procedure che sono definite successivamente nel modulo, o addirittura in un altro modulo.

Esempi

L'esempio successivo presenta un elenco vuoto di parametri nell'istruzione **DECLARE**, dal momento che nessun argomento viene passato a `OttInput$`:

```
DECLARE FUNCTION OttInput$()
X$ = OttInput$

FUNCTION OttInput$ STATIC
   OttInput$ = INPUT$(10)
END FUNCTION
```

L'esempio successivo presenta un elenco di parametri nell'istruzione **DECLARE**, perché un argomento intero viene passato a questa versione di `OttInput$`:

```
DECLARE FUNCTION OttInput$(X%)
X% = OttInput$(5)

FUNCTION OttInput$(X%) STATIC
   OttInput$ = INPUT$(X%)
END FUNCTION
```

2.5.4.1 Quando QuickBASIC non genera l'istruzione DECLARE

Esistono dei casi in cui QuickBASIC non produce le istruzioni **DECLARE** nel modulo che chiama una procedura.

In primo luogo, QuickBASIC non può generare un'istruzione **DECLARE** in un modulo che chiama una procedura **FUNCTION** definita in un altro modulo, se quest'ultimo non è caricato in memoria.

In questo caso l'utente stesso deve digitare l'istruzione **DECLARE** all'inizio del modulo dove viene chiamata la **FUNCTION**; altrimenti, QuickBASIC confonderà la chiamata della **FUNCTION** con il nome di una variabile.

In secondo luogo, QuickBASIC non produce un'istruzione **DECLARE** per la chiamata di una procedura **SUB** in un altro modulo, sia che il modulo sia caricato o meno. L'istruzione **DECLARE** non è necessaria, però, a meno che non si voglia chiamare la procedura **SUB** senza utilizzare la parola chiave **CALL**.

Infine, QuickBASIC non può produrre un'istruzione **DECLARE** per alcuna procedura in una libreria Quick. L'utente stesso dovrà provvedere ad aggiungere tale istruzione.

2.5.4.2 Sviluppo di programmi fuori dell'ambiente QuickBASIC

Se l'utente scrive dei programmi con il proprio editor di testo e poi li compila fuori dell'ambiente QuickBASIC con i comandi BC e LINK, deve assicurarsi di inserire istruzioni **DECLARE** nelle tre posizioni seguenti:

- All'inizio di ogni modulo che chiama una procedura **FUNCTION** prima che essa sia definita:

```
DECLARE FUNCTION Ipot (X!, Y!)

INPUT X, Y
PRINT Ipot (X, Y)
END

FUNCTION Ipot (A, B) STATIC
   Ipot = SQR (A ^ 2 + B ^ 2)
END FUNCTION
```

- All'inizio di ogni modulo che chiama una procedura **SUB** prima che essa sia definita, e che non utilizza **CALL** per chiamare la procedura:

```
DECLARE SUB StampaStr (X, Y)
INPUT X, Y

StampaStr X, Y        ' Nota: gli argomenti non sono
                      ' racchiusi in parentesi.
END
```

```
SUB StampaStr (A, B) STATIC

   ' Converte i numeri in stringhe, rimuove gli spazi
   ' iniziali dal secondo numero, e stampa:
   PRINT STR$(A) + LTRIM$(STR$(B))
END SUB
```

Quando si chiama una procedura **SUB** con **CALL**, non è necessario dichiararla prima:

```
A$ = "466"
B$ = "123"
CALL StampaStr (A$, B$)
END

SUB StampaStr (X$, Y$) STATIC
   PRINT VAL(X$) + VAL(Y$)
END SUB
```

- All'inizio di qualsiasi modulo che chiama una procedura **SUB** o **FUNCTION** definita in un altro modulo (una "procedura esterna").

Se la procedura non ha parametri, bisogna far seguire da parentesi vuote il nome della procedura in un'istruzione **DECLARE**, come nell'esempio successivo:

```
DECLARE FUNCTION OttOra$()
PRINT OttOra$
END

FUNCTION OttOra$ STATIC
   OttOra$ = LEFT$(TIME$,2)
END FUNCTION
```

E' necessario ricordare che un'istruzione **DECLARE** può comparire solo a livello di modulo, non a livello di procedura; e che influisce sull'intero modulo in cui essa compare.

2.5.4.3 Utilizzo dei file da includere per dichiarazioni

Quando si crea un modulo a parte per la definizione di una o più procedure **SUB** o **FUNCTION**, è opportuno creare anche un file da includere relativo a questo modulo. Questo file da includere dovrebbe contenere quanto segue:

- Istruzioni **DECLARE** per tutte le procedure del modulo.

continua

- Definizioni **TYPE...END TYPE** di record per tutti i parametri a record nelle procedure **SUB** o **FUNCTION** di questo modulo.
- Istruzioni **COMMON** che elenchino le variabili condivise tra questo modulo e altri moduli del programma. (Per ulteriori informazioni sull'utilizzo di **COMMON**, consultare la sezione 2.6.3, "Condivisione di variabili con altri moduli".)

Ogni volta che si utilizza il modulo di definizioni in uno dei programmi, inserire un metacomando **$INCLUDE** all'inizio di qualsiasi modulo che richiama procedure nel modulo di definizioni. Quando poi il programma viene compilato, il contenuto del file da includere verrà sostituito al metacomando **$INCLUDE**.

E' consigliabile creare un file da includere per ogni modulo e poi utilizzare insieme il modulo ed il file da includere come delineato sopra. L'elenco seguente presenta dettagliatamente alcuni vantaggi di questa tecnica:

- Il modulo contenente definizioni di procedure resta realmente modulare – non è necessario copiarne l'istruzione **DECLARE** per ogni chiamata di una sua procedura da parte di un altro modulo; si può invece utilizzare un solo metacomando **$INCLUDE**.
- L'utilizzazione di un file da includere per le dichiarazioni delle procedure evita la produzione automatica delle istruzioni **DECLARE** durante la memorizzazione di un programma nell'ambiente QuickBASIC.
- L'utilizzo di file da includere per le dichiarazioni elimina eventuali problemi nel riconoscimento da parte di un modulo di funzioni in un altro modulo. (Per ulteriori informazioni, consultare la sezione 2.5.4.1, "Quando QuickBASIC non genera l'istruzione DECLARE".)

La facilità con cui QuickBASIC produce le istruzioni **DECLARE** risulta estremamente vantaggiosa quando si crea un file da includere. I punti seguenti spiegano le modalità dell'operazione:

1. Creare il modulo.
2. Chiamare, dall'interno di quel modulo, qualunque procedura **SUB** o **FUNCTION** che sia stata definita.
3. Memorizzare il modulo per ottenere automaticamente istruzioni **DECLARE** per tutte le procedure.
4. Modificare nuovamente il modulo: togliere le chiamate alle procedure e spostare le istruzioni **DECLARE** in un file da includere separato.

Per ulteriori informazioni relative alla sintassi e all'utilizzo del metacomando **$INCLUDE**, consultare l'appendice F, "Metacomandi".

Esempio

I frammenti seguenti illustrano come utilizzare insieme un modulo di definizioni ed un file da includere:

```
' ===========================================================
'                         MODDEF.BAS
'  Questo modulo contiene le definizioni per le procedure
'  Prompter e Mass!
' ===========================================================

FUNCTION Mass! (X!, Y!) STATIC
   IF X! > Y! THEN Mass! = X! ELSE Mass! = Y!
END FUNCTION

SUB Prompter (Riga%, Colonna%, VarRec AS TipoRec) STATIC
   LOCATE Riga%, Colonna%
   INPUT "Descrizione: ", VarRec.Descrizione
   INPUT "Quantità:    ", VarRec.Quantita
END SUB

' ===========================================================
'                         MODDEF.BI
'  Questo è un file da includere che contiene le istruzioni
'  DECLARE per le procedure Prompter e Mass! (così come
'  un'istruzione TYPE che definisce il tipo di dati
'  TipoRec). Utilizzare questo file dovunque viene
'  utilizzato il modulo MODDEF.BAS.
' ===========================================================

TYPE TipoRec
   Descrizione AS STRING * 15
   Quantita AS INTEGER
END TYPE

DECLARE FUNCTION Mass! (X!, Y!)
DECLARE SUB Prompter (Riga%, Colonna%, VarRec AS TipoRec)

' ===========================================================
'                        SAMPLE.BAS
'  Questo modulo è linkato al modulo MODDEF.BAS, e
'  chiama le procedure Prompter e Mass! in MODDEF.BAS.
' ===========================================================
```

```
' La riga successiva rende il contenuto del file da
' includere MODDEF.BI parte di questo modulo:
' $INCLUDE: 'MODDEF.BI'
.
.
.
INPUT A, B
PRINT Mass!(A, B)          ' Chiama la FUNCTION Mass! nel
.                          ' file MODDEF.BAS
.
.
Prompter 5, 5, VarRec      ' Chiama la SUB Prompter nel
.                          ' file MODDEF.BAS
.
.
```

Importante E' buona norma di programmazione inserire in un file da includere le dichiarazioni di una procedura, ma non sono da inserire i blocchi **SUB...END SUB** o **FUNCTION...END FUNCTION**: nella versione 4.5 di QuickBASIC non è permesso infatti inserire nei file da includere le definizioni delle procedure. Se sono stati utilizzati file da includere per definire delle procedure **SUB** in programmi scritti con la versione 2.0 o 3.0 di QuickBASIC, bisogna inserire queste definizioni in un modulo a parte, oppure incorporarle nel modulo dal quale sono chiamate.

2.5.4.4 Dichiarazione di procedure nelle librerie Quick

Un buon metodo di programmazione consiste nell'inserire in un file da includere tutte le dichiarazioni delle procedure contenute in una libreria Quick. Con il metacomando **$INCLUDE**, questo file da includere si può poi incorporare nei programmi che utilizzano la libreria. Ciò evita di dover copiare tutte le relative istruzioni **DECLARE** ogni volta che si utilizza la libreria.

2.5.5 Passaggio di argomenti per riferimento

Le variabili – sia le semplici variabili scalari che le matrici e gli elementi di matrice o i record – per predefinizione vengono passate alle procedure **FUNCTION** e **SUB** "per riferimento". Passare le variabili per riferimento significa che:

- Ciascuna variabile del programma ha un indirizzo, o posizione in memoria, corrispondente al punto dove viene memorizzato il suo valore.
- La procedura chiamata riceve l'indirizzo di ciascuna variabile, per cui l'indirizzo della variabile e del suo corrispondente parametro all'interno della procedura sono identici.

- Di conseguenza, se la procedura modifica il valore del parametro, modificherà anche il valore della variabile che le viene passata.

Se non si desidera che una procedura modifichi il valore di una variabile, bisogna passarle il valore contenuto nella variabile piuttosto che il suo indirizzo. In questo modo viene modificata solo una copia della variabile, e non la variabile stessa. La sezione successiva descrive questo metodo alternativo per il passaggio delle variabili.

Esempio

Nel programma che segue, le modifiche eseguite sul parametro `A$` nella procedura `Scambia` modificano anche l'argomento `Test$`:

```
Test$ = "una stringa con tutte le lettere minuscole."
PRINT "Prima della chiamata del sottoprogramma: "; Test$
CALL Scambia (Test$, "a")
PRINT "Dopo la chiamata del sottoprogramma: "; Test$
END

SUB Scambia (A$, B$) STATIC
   Inizio = 1
   DO

      ' Cerca B$ all'interno di A$, cominciando dal
      ' carattere di A$ con posizione "Inizio":
      Trovato = INSTR (Inizio, A$, B$)
      ' Scambia ogni istanza di B$ in A$ con
      ' lettere maiuscole:
      IF Trovato > 0 THEN
         MID$(A$, Trovato) = UCASE$(B$)
         Inizio = Inizio + 1
      END IF
   LOOP WHILE Trovato > 0
END SUB
```

Output

```
Prima della chiamata del sottoprogramma: una stringa con
tutte le lettere minuscole.
Dopo la chiamata del sottoprogramma: unA stringA con tutte
le lettere minuscole.
```

2.5.6 Passaggio di argomenti per valore

Passare un argomento "per valore", significa passare il valore dell'argomento, piuttosto che il suo indirizzo. Nelle procedure BASIC, il passaggio di una variabile per valore avviene copiando la variabile ad una posizione temporanea, quindi passando l'indirizzo di tale posizione. Dato che la procedura non ha accesso all'indirizzo della variabile originale, non può modificarla; apporta invece tutte le eventuali modifiche alla copia.

Alle procedure possono essere passate espressioni come argomento, come mostrato in seguito:

```
' A + B è un'espressione; i valori di A e di B non sono
' influenzati dalla chiamata di questa procedura:
CALL Mult (A + B, B)
```

Le espressioni vengono sempre passate per valore. (Per ulteriori informazioni, consultare la sezione 2.5.2, "Passaggio di costanti ed espressioni".)

Esempio

Un modo per passare una variabile in base al valore consiste nel trasformarla in un'espressione, racchiudendola in parentesi. Come si può vedere dall'output che segue, le modifiche apportate alla variabile locale Y nella procedura **SUB** vengono restituite al livello del modulo come modifiche apportate alla variabile B. Tuttavia, le modifiche della variabile X nella procedura non influiscono sul valore di A, dal momento che A viene passata per valore.

```
A = 1
B = 1
PRINT "Prima della chiamata al sottoprogramma, A ="; A; ", _
   B ="; B

' A viene passata per valore, mentre B viene passato per
' riferimento:
CALL Mult ((A), B)
PRINT "Dopo la chiamata al sottoprogramma, A ="; A; ", _
   B ="; B
END

SUB Mult (X, Y) STATIC
   X = 2 * X
   Y = 3 * Y
   PRINT "Nel sottoprogramma, X ="; X; ", Y ="; Y
END SUB
```

Output

```
Prima della chiamata al sottoprogramma, A = 1, B = 1
Nel sottoprogramma, X = 2, Y = 3
Dopo la chiamata al sottoprogramma, A = 1, B = 3
```

2.6 Variabili condivise con SHARED

Oltre a passare le variabili attraverso argomenti ed elenchi di parametri, le procedure possono anche condividere le variabili con altre procedure e con il programma a livello di modulo (cioè, il programma all'interno di un modulo ma fuori da qualsiasi procedura) in uno dei due seguenti modi:

- Le variabili elencate in un'istruzione **SHARED** all'interno di una procedura sono condivise soltanto tra quella procedura e il programma a livello di modulo. Questo metodo viene utilizzato quando le diverse procedure di uno stesso modulo hanno bisogno di diverse combinazioni delle variabili a livello di modulo.
- Le variabili elencate in un'istruzione **COMMON SHARED**, **DIM SHARED**, o **REDIM SHARED** a livello di modulo vengono condivise tra il programma a livello di modulo e tutte le procedure all'interno di quel modulo. Questo metodo è utilizzato per lo più quando tutte le procedure di un modulo utilizzano un'insieme comune di variabili.

Si possono anche utilizzare le istruzioni **COMMON** o **COMMON SHARED** per condividere variabili tra due o più moduli. Le sezioni 2.6.1–2.6.3 trattano questi tre metodi di condivisione delle variabili.

2.6.1 Variabili condivise con procedure specifiche in un modulo

Se procedure diverse all'interno di un modulo devono condividere variabili diverse con il programma a livello di modulo, utilizzare un'istruzione **SHARED** all'interno di ciascuna procedura.

I nomi di matrice appaiono nelle istruzioni **SHARED** seguite da parentesi vuote ( ):

```
SUB UnAltraSub STATIC
   SHARED NomeDiMatrice()
   .
   .
   .
```

In un'istruzione **SHARED** un tipo di variabile dichiarato con la clausola **AS** *tipo* deve essere digitato allo stesso modo:

```
DIM Buffer AS STRING * 10
.
.
.
END

SUB LeggiRecord STATIC
   SHARED Buffer AS STRING * 10
   .
   .
   .
END SUB
```

Esempio

Nell'esempio successivo, le istruzioni **SHARED** nelle procedure `OttRecord` e `InventarioTotale` mostrano il formato di un elenco di variabili condivise:

```
' ==========================================================
'             Programma a livello di modulo
' ==========================================================

TYPE TipoRec
   Prezzo AS SINGLE
   Desc AS STRING * 35
END TYPE

DIM VarRec (1 TO 100) AS TipoRec      ' Matrice di record

INPUT "Nome del file:", FileSpec$
CALL OttRecord
PRINT InventarioTotale
END

' ==========================================================
'             Programma a livello di procedura
' ==========================================================
```

```
SUB OttRecord STATIC

' Sia FileSpec$ che la matrice di record VarRec è
' condivisa con il precedente programma a livello di modulo:
   SHARED FileSpec$, VarRec() AS TipoRec
   OPEN FileSpec$ FOR RANDOM AS #1
    .
    .
    .
END SUB

FUNCTION InventarioTotale STATIC

' Soltanto la matrice VarRec è condivisa con il programma
' a livello di modulo:
   SHARED VarRec() AS TipoRec
    .
    .
    .
END FUNCTION
```

2.6.2 Condivisione di variabili con tutte le procedure di un modulo

Se delle variabili vengono dichiarate con l'attributo **SHARED** in un'istruzione **COMMON**, **DIM**, o **REDIM** (per esempio utilizzando un'istruzione **COMMON SHARED** *elencovariabili*) a livello del modulo, allora tutte le procedure che si trovano all'interno di quel modulo possono accedere a quelle variabili. In altri termini l'attributo **SHARED** rende le variabili disponibili in ogni parte del modulo.

E' opportuno utilizzare l'attributo **SHARED** quando si deve condividere un gran numero di variabili con tutte le procedure di un modulo.

Esempi

Queste istruzioni dichiarano le variabili condivise tra tutte le procedure del modulo:

```
COMMON SHARED A, B, C
DIM SHARED Matrice(1 TO 10, 1 TO 10) AS TipoUtente
REDIM SHARED Alfa(N%)
```

Nell'esempio seguente, il programma a livello di modulo condivide la matrice a stringhe `MatrStr` e le variabili intere `Min` e `Mass` con le due procedure **SUB** `RiempiMatr` e `StampaMatr`:

```
' ==========================================================
'              Programma a livello di modulo
' ==========================================================

DECLARE SUB RiempiMatr()
DECLARE SUB StampaMatr()

' Le seguenti istruzioni DIM mettono in comune le variabili
' intere Min e Mass e la matrice a stringhe MatrStr con
' tutte le SUB e FUNCTION di questo modulo:
DIM SHARED MatrStr (33 TO 126) AS STRING * 5
DIM SHARED Min AS INTEGER, Mass AS INTEGER

Min = LBOUND(MatrStr)
Mass = UBOUND(MatrStr)

RiempiMatr    ' Notare l'assenza degli elenchi di argomenti.
StampaMatr
END

' ==========================================================
'             Programma a livello di procedura
' ==========================================================

SUB RiempiMatr STATIC

   ' Riempie ciascun elemento della matrice da 33 a 126 con
   ' una stringa di 5 caratteri, ciascuno con il codice
   ' ASCII uguale a I%:
   FOR I% = Min TO Mass
      MatrStr(I%) = STRING$(5, I%)
   NEXT

END SUB

SUB StampaMatr STATIC
   FOR I% = Min TO Mass
      PRINT MatrStr(I%)
   NEXT
END SUB
```

Output parziale

```
!!!!!
"""""
#####
$$$$$
%%%%%
&&&&&
'''''
(((((
.
.
.
```

Se l'utente scrive dei programmi con il proprio editor di testo e poi li compila completamente fuori dell'ambiente di QuickBASIC, noterà che le variabili dichiarate con **SHARED** devono precedere la definizione delle procedure. In caso contrario, le procedure non potranno disporre del valore delle variabili **SHARED**, come dimostra l'output nell'esempio seguente. (In QuickBASIC, questa sequenza non è necessaria, perché i programmi vengono memorizzati automaticamente nell'ordine esatto.)

```
DEFINT A-Z

FUNCTION Somma (X, Y) STATIC
   Somma = X + Y + Z
END FUNCTION

DIM SHARED Z
Z = 2
PRINT Somma (1, 3)
END
```

Output

```
 4
```

L'esempio successivo mostra come salvare il modulo precedente, definendo `Somma` successivamente all'istruzione `DIM SHARED Z`:

```
DEFINT A-Z

DECLARE FUNCTION Somma (X, Y)
```

continua

```
' La variabile Z ora viene condivisa con Somma:
DIM SHARED Z
Z = 2
PRINT Somma (1, 3)
END

FUNCTION Somma (X, Y) STATIC
   Somma = X + Y + Z
END FUNCTION
```

Output

```
 6
```

2.6.3 Condivisione di variabili con altri moduli

Se si vogliono condividere delle variabili con altri moduli del programma, è necessario elencare le variabili in istruzioni **COMMON** o **COMMON SHARED** a livello di modulo in ciascun modulo.

Esempi

L'esempio seguente mostra come condividere variabili tra moduli diversi utilizzando un'istruzione **COMMON** nel modulo che chiama le procedure **SUB**, e un'istruzione **COMMON SHARED** nel modulo che definisce le procedure. Con **COMMON SHARED**, tutte le procedure del secondo modulo possono accedere alle variabili in comune:

```
' ===========================================================
'                    Modulo Principale
' ===========================================================

COMMON A, B
A = 2.5
B = 1.2
CALL Quadrato
CALL Cubo
END
```

```
' ==========================================================
'           Modulo con le procedure Cubo e Quadrato
' ==========================================================

' Nota: I nomi delle variabili (X e Y) possono essere
' diversi che nell'altro modulo (A e B); solo i tipi
' devono essere uguali.
COMMON SHARED X, Y        ' Questa istruzione è a livello di
                          ' modulo. X e Y sono condivise con
                          ' le procedure Cubo e Quadrato
                          ' seguenti.
SUB Cubo STATIC
   PRINT "A al cubo è uguale a  "; X ^ 3
   PRINT "B al cubo è uguale a  "; Y ^ 3
END SUB

SUB Quadrato STATIC
   PRINT "A al quadrato è uguale a  "; X ^ 2
   PRINT "B al quadrato è uguale a  "; Y ^ 2
END SUB
```

L'esempio successivo utilizza i blocchi **COMMON** al livello dei moduli e le istruzioni **SHARED** all'interno delle procedure per condividere contemporaneamente un diverso insieme di variabili con ogni procedura:

```
' ==========================================================
'                    Modulo Principale
'  Stampa il volume e la densità di un cilindro solido
'  a seconda dei dati digitati.
' ==========================================================

COMMON /ValoriVol/ Altezza, Raggio, Volume
COMMON /ValoriDensita/ Peso, Densita

INPUT "Altezza del cilindro (in cm) : ", Altezza
INPUT "Raggio del cilindro (in cm) : ", Raggio
INPUT "Peso del cilindro (in g) : ", Peso

CALL CalcVolume
CALL CalcDensita
```

```
PRINT "Il volume è di"; Volume; "centimetri cubici."
PRINT "La densità è di"; Densita; "grammi per ";
PRINT "centimetro cubico."
END

' ===========================================================
'     Modulo con le procedure CalcVolume e CalcDensita
' ===========================================================

COMMON /ValoriVol/ A, R, V
COMMON /ValoriDensita/ P, D

SUB CalcVolume STATIC

   ' Condivide le variabili dell'altezzza, raggio e volume
   ' con questa procedura:
   SHARED A, R, V
   CONST PIGRECO = 3.141592653589#
   V = PIGRECO * A * (R ^ 2)
END SUB

SUB CalcDensita

   ' Condivide le variabili del peso, volume e densità
   ' con questa procedura:
   SHARED P, V, D
   D = P / V
END SUB
```

Output

```
Altezza del cilindro (in cm) : 100
Raggio del cilindro (in cm) : 10
Peso del cilindro (in g) : 10000
Il volume è di 31415.93 centimetri cubici.
La densità è di .3183099 grammi per centimetro cubico.
```

2.6.4 Il problema delle variabili sinonime

Nei programmi molto lunghi che contengono un gran numero di variabili e di procedure, può verificarsi il caso delle "variabili sinonime", in cui due o più nomi si riferiscono alla stessa posizione nello spazio memoria. Ciò si verifica:

- Quando la stessa variabile compare più di una volta nell'elenco degli argomenti passati ad una procedura.
- Quando una variabile viene passata ad una procedura in un'elenco di argomenti e anche per mezzo di un'istruzione **SHARED** o di un attributo **SHARED**.

Per evitare problemi di sinonimia, è opportuno effettuare un rigoroso controllo delle variabili condivise con una procedura per evitare che tali variabili siano presenti anche nell'elenco degli argomenti di chiamata della procedura. Non bisogna inoltre passare due volte la stessa variabile, come nell'istruzione successiva:

```
' X viene passato due volte, per cui sorgeranno problemi di
' sinonimia nella procedura Test:
CALL Test (X, X, Y)
```

Esempio

L'esempio successivo mostra un caso di sinonimia tra variabili: la variabile `A` viene condivisa tra il programma a livello di modulo e la procedura **SUB** per mezzo dell'istruzione `DIM SHARED`. Inoltre, `A` viene passata per riferimento alla procedura **SUB** come argomento. Di conseguenza, nel sottoprogramma, `A` ed `X` si riferiscono alla stessa posizione nello spazio memoria, e quando il sottoprogramma modifica `X`, viene modificata anche `A` e viceversa.

```
DIM SHARED A
A = 4
CALL StampaMeta
END

SUB StampaMeta (X) STATIC
   PRINT "Metà di"; X; "più metà di"; A; "è uguale a";
   X = X / 2      ' X e A ora sono entrambe uguali a 2.
   A = A / 2      ' X e A ora sono entrambe uguali a 1.
   PRINT A + X
END SUB
```

Output

```
Metà di 4 più metà di 4 è uguale a 2
```

2.7 Variabili automatiche e variabili STATIC

L'attributo **STATIC** nella definizione di una procedura indica che le variabili locali all'interno della procedura sono **STATIC**: i loro valori rimangono inalterati tra una chiamata alla procedura e l'altra.

Omettendo l'attributo **STATIC**, le variabili locali all'interno della procedura diventano automatiche per predefinizione; cioè, si ottiene un nuovo insieme di variabili locali ogni volta che viene chiamata la procedura

E' possibile tuttavia ovviare all'omissione dell'istruzione **STATIC** nella definizione, utilizzando un'istruzione **STATIC** all'interno della procedura; di conseguenza alcune variabili saranno automatiche ed altre **STATIC**. (Per ulteriori informazioni, consultare la sezione 2.8.)

Nota L'istruzione **SHARED** ha la precedenza sulla predefinizione delle variabili di una procedura come locali **STATIC** o locali automatiche, perché qualunque variabile che si presente in un'istruzione **SHARED** è riconosciuta a livello di modulo e quindi non è locale rispetto alla procedura.

2.8 Conservazione dei valori delle variabili locali con l'istruzione STATIC

Talvolta è desiderabile, all'interno di una procedura, rendere alcune variabili locali **STATIC** lasciando le altre automatiche; elencare le prime in un'istruzione **STATIC** all'interno della procedura. Questo è anche un metodo per assicurare che una variabile sia locale, perché l'istruzione **STATIC** avrà la precedenza su eventuali istruzioni **SHARED** a livello di modulo.

Un'istruzione **STATIC** può comparire soltanto all'interno di una procedura. Il nome di una matrice in un'istruzione **STATIC** deve essere seguito da una coppia di parentesi vuote ( ) e la matrice deve essere dimensionata prima di poter essere utilizzata, come mostra l'esempio successivo:

```
SUB SottoProg2
   STATIC Matrice() AS INTEGER
   DIM Matrice(-5 TO 5, 1 TO 25) AS INTEGER
   .
   .
   .
END SUB
```

Nota Se si definisce il tipo di una variabile con la clausola **AS** *tipo*, questa clausola deve apparire in entrambe le istruzioni **STATIC** e **DIM**.

Esempio

L'esempio seguente mostra il modo in cui un'istruzione **STATIC** conserva il valore della variabile a stringa `Y$` tra una chiamata a `TestSub` e l'altra:

```
DECLARE SUB TestSub()
FOR I% = 1 TO 5
   TestSub         ' Chiama TestSub 5 volte
NEXT I%
END

SUB TestSub        ' Nota: nessun attributo STATIC

   ' Entrambe le variabili X$ e Y$ sono locali in TestSub,
   ' cioè i loro valori non sono condivisi con il programma
   ' a livello di modulo. Dato che X$ è automatica, viene
   ' reinizializzata alla stringa nulla a ogni chiamata di
   ' TestSub. Y$, però, essendo STATIC, mantiene il valore
   ' della chiamata precedente:
   STATIC Y$
   X$ = X$ + "*"
   Y$ = Y$ + "*"
   PRINT X$, Y$
END SUB
```

Output

```
*       *
*       **
*       ***
*       ****
*       *****
```

2.9 Procedure ricorsive

Nel BASIC le procedure possono essere ricorsive. Una procedura di tipo ricorsivo può effettuare una chiamata a se stessa o ad altre procedure che successivamente chiamano la prima.

2.9.1 La funzione fattoriale

Un metodo efficace per illustrare le procedure ricorsive è quello di prendere in considerazione la funzione fattoriale. n! (n fattoriale) può essere definito con la seguente formula:

$n! = n * (n - 1) * (n - 2) * ... * 2 * 1$

Ad esempio, 5 fattoriale viene valutato come segue:

$5! = 5 * 4 * 3 * 2 * 1 = 120$

Nota Non bisogna confondere il simbolo fattoriale matematico (!) utilizzato qui con il suffisso di dichiarazione del tipo dati in precisione semplice utilizzato dal BASIC.

I fattoriali stessi si adattano ad una definizione ricorsiva:

$n! = n * (n - 1)!$

Ciò porta alla seguente progressione:

$5! = 5 * 4!$

$4! = 4 * 3!$

$3! = 3 * 2!$

$2! = 2 * 1!$

$1! = 1 * 0!$

Una ricursione deve sempre avere una condizione limite. Con i fattoriali questa condizione limite si verifica quando si valuta 0!, che ha, per definizione, il valore 1.

Nota Sebbene una procedura ricorsiva possa avere delle variabili **STATIC** per predefinizione (come nell'esempio successivo), è spesso preferibile lasciare che le variabili predefinite siano automatiche. In questo modo, le chiamate ricorsive non sovrascriveranno i valori delle variabili di una precedente chiamata.

Esempio

Nell'esempio seguente viene utilizzata una procedura **FUNCTION** ricorsiva per il calcolo di numeri fattoriali:

```
DECLARE FUNCTION Fattoriale#(N%)
Formato$ = "###_! = ###################"
DO
   INPUT "Digitare un numero tra 0 e 20 (o -1 per _
      uscire): ", Num%
   IF Num% >= 0 AND Num% <= 20 THEN
      PRINT USING Formato$; Num%; Fattoriale#(Num%)
   END IF
LOOP WHILE Num% >= 0
END

FUNCTION Fattoriale#(N%) STATIC

   IF N% > 0 THEN         ' Chiama Fattoriale# di nuovo se
                          ' N è maggiore di zero.
      Fattoriale# = N% * Fattoriale#(N - 1)

   ELSE                   ' Ha raggiunto la condizione limite
                          ' (N% = 0), e ritorna al livello
                          ' precedente.
      Fattoriale# = 1
   END IF
END FUNCTION
```

2.9.2 Modifica della dimensione dello stack

La ricursione può consumare una grande quantità di memoria, perché ciascun gruppo di variabili automatiche di una procedura **SUB** o **FUNCTION** viene salvato nello stack. (Questo tipo di memorizzazione permette ad una procedura di conservare i valori delle variabili durante una chiamata ricorsiva.)

Se si ha una procedura ricorsiva con molte variabili automatiche, oppure una procedura ricorsiva molto nidificata, può essere necessario modificare la dimensione dello stack con un'istruzione **CLEAR, ,** *dimensionestack*, in cui *dimensionestack* corrisponde al numero di byte da assegnare allo stack. In caso contrario, durante l'esecuzione del programma, si può ottenere il messaggio di errore `Spazio stack esaurito`.

I seguenti punti indicano come stimare la quantità di memoria necessaria ad una procedura ricorsiva:

1. Inserire una sola virgoletta per trasformare temporaneamente la chiamata ricorsiva in riga di commento, in modo da far chiamare la procedura soltanto una volta durante l'esecuzione del programma.
2. Chiamare la funzione **FRE(−2)** (che restituisce la quantità di spazio inutilizzato dello stack) prima di chiamare la procedura ricorsiva e richiamarla alla fine della procedura. Utilizzare le istruzioni **PRINT** per visualizzare i valori riportati.
3. Eseguire il programma. La differenza dei valori rappresenta la quantità di spazio dello stack (in byte) utilizzata da una chiamata della procedura.
4. Stimare il numero massimo di volte che la procedura sarà chiamata, quindi moltiplicare questo valore per lo spazio dello stack utilizzato da una singola chiamata alla procedura. Il risultato dà il *totalebyte*.
5. Assegnare allo stack la quantità di spazio calcolata nel punto 4:

   CLEAR,, *totalebyte*

2.10 Trasferimento del controllo ad un altro programma con CHAIN

A differenza delle chiamate di procedura, che si verificano all'interno dello stesso programma, l'istruzione **CHAIN** dà inizio ad un altro programma. Quando il programma si concatena ad un altro, si verifica la seguente sequenza di azioni:

1. Si arresta l'esecuzione del primo programma.
2. Il secondo programma viene caricato in memoria.
3. Inizia l'esecuzione del secondo programma.

L'istruzione **CHAIN** dà la possibilità di dividere in diversi programmi più piccoli un programma con ampi requisiti di memoria.

L'istruzione **COMMON** permette il passaggio delle variabili tra due o più programmi concatenati.

Una buona norma di programmazione consiste nell'inserire le istruzioni **COMMON** in un file da includere e quindi di utilizzare il metacomando **$INCLUDE** all'inizio di ciascun programma concatenato.

Nota Non utilizzare un'istruzione **COMMON** /*nomeblocco*/*elencovariabili* (un "blocco denominato **COMMON**") per passare le variabili ad un programma concatenato, perché le variabili elencate in blocchi di istruzioni **COMMON** non vengono conservate nella concatenazione. Utilizzare invece un blocco d'istruzioni **COMMON** vuoto (**COMMON** *elencovariabili*).

Esempio

In quest'esempio, in cui viene presentata una concatenazione di tre programmi diversi, viene utilizzato un file da includere per dichiarare le variabili comuni ai programmi:

```
' ====== Il contenuto del file da includere COMUNE.BI ======

DIM Valori(10)
COMMON Valori(), NumVal


' ====================== PRINC.BAS ========================

' Copia il contenuto del file COMUNE.BI:
' $INCLUDE: 'COMUNE.BI'

   ' Legge i dati:
   INPUT "Digitare il numero dei valori (<= 10): ", NumVal
   FOR I = 1 TO NumVal
      Prompt$ = "Valore ("+LTRIM$(STR$(I))+")? "
      PRINT Prompt$;
      INPUT "", Valori(I)
   NEXT I

   ' Chiede all'utente il calcolo da eseguire:
   INPUT "Calcolo (1 = dispersione, 2 = media)? ", Scelta

   ' Ora si concatena al programma scelto:
   SELECT CASE Scelta
      CASE 1:                  ' Dispersione
         CHAIN "DISP"
      CASE 2:                  ' Media
         CHAIN "MEDIA"
   END SELECT
END
```

```
' ====================== DISP.BAS ========================
'  Calcola la dispersione di un insieme di dati.
' ========================================================

' $INCLUDE: 'COMUNE.BI'

   Somma = 0    ' Somma dei valori
   SommaQ = 0   ' Somma dei valori al quadrato

   FOR I = 1 TO NumVal
      Somma = Somma + Valori(I)
      SommaQ = SommaQ + Valori(I) ^ 2
   NEXT I

   Disp = SQR(SommaQ / NumVal - (Somma / NumVal) ^ 2)
   PRINT "La dispersione del campione è: " Disp
END

' ======================= MEDIA.BAS ======================
'  Calcola la media di un insieme di dati.
' ========================================================

' $INCLUDE: 'COMUNE.BI'

   Somma = 0

   FOR I = 1 TO NumVal
      Somma = Somma + Valori(I)
   NEXT

   Media = Somma / NumVal
   PRINT "La media del campione è: " Media
END
```

2.11 Esempio di applicativo: ricerca ricorsiva nelle directory (DOVE.BAS)

Nel seguente programma si utilizza una procedura ricorsiva **SUB**, `GuardaDir`, per ricercare su disco il nome di un file digitato dall'utente. Ogni volta che questo programma trova il file chiesto, ne visualizza il percorso completo.

Istruzioni e funzioni utilizzate

Il programma listato dimostra l'utilizzo delle seguenti istruzioni e parole chiave discusse nel presente capitolo:

- **DECLARE**
- **FUNCTION...END FUNCTION**
- **STATIC**
- **SUB...END SUB**

Listato del programma

```
DEFINT A-Z

' Dichiara le costanti simboliche usate nel programma:
CONST TIPOEOF = 0, TIPOFILE = 1, TIPODIR = 2, RADICE = "TWH"

DECLARE SUB GuardaDir (SpecPerc$, Livello, SpecFile$, Riga)

DECLARE FUNCTION CreaNomeFile$ (Num)
DECLARE FUNCTION OttVoce$ (NumFile, TipoVoce)

CLS
INPUT "File da trovare"; SpecFile$
PRINT
PRINT "Digitare la directory da cui iniziare la ricerca "
PRINT "(l'unità disco (opzionale) + la directory), o "
PRINT "premere <INVIO> per iniziare la ricerca dalla "
PRINT "directory principale dell'unità corrente."
PRINT
INPUT "Directory iniziale"; SpecPerc$
CLS
```

```
CarDes$ = RIGHT$(SpecPerc$, 1)

IF SpecPerc$ = "" OR CarDes$ = ":" OR CarDes$ <> "\" THEN
   SpecPerc$ = SpecPerc$ + "\"
END IF

SpecFile$ = UCASE$(SpecFile$)
SpecPerc$ = UCASE$(SpecPerc$)
Livello = 1
Riga = 3

' Esegue la chiamata al livello più alto (il livello 1)
' per iniziare la ricerca:
GuardaDir SpecPerc$, Livello, SpecFile$, Riga

KILL RADICE + ".*"          ' Elimina tutti i file temporanei
                            ' creati dal programma.

LOCATE Riga + 1, 1: PRINT "Ricerca compiuta."
END

' ===================== CREANOMEFILE$ ======================
'  Questa procedura crea un nome di file da una radice
'  ("TWH" -- definita come costante simbolica al livello
'  del modulo) e un numero passato come argomento (Num).
' ==========================================================

FUNCTION CreaNomeFile$(Num) STATIC

   CreaNomeFile$ = RADICE + "." + LTRIM$(STR$(Num))

END FUNCTION

' ======================= GUARDADIR ========================
'  Questa procedura ricerca ricorsivamente in una directory
'  il file con il nome digitato dall'utente.
'
'  NOTA: La testata della SUB non usa la parola chiave
'        STATIC perché la procedura usa nuove variabili
'        ad ogni chiamata.
' ==========================================================

SUB GuardaDir (SpecPerc$, Livello, SpecFile$, Riga)
```

```
LOCATE 1, 1: PRINT "Ricerca in corso in "; SPACE$(50);
LOCATE 1, 21: PRINT SpecPerc$;

' Crea un nome per il file temporaneo:
FileTemp$ = CreaNomeFile$(Livello)

' Ottiene un listato della directory corrente,
' e lo salva nel file temporaneo:
SHELL "DIR " + SpecPerc$ + " > " + FileTemp$

' Trova il primo numero di file disponibile:
NumFile = FREEFILE

' Apre il file con il listato DIR e lo legge:
OPEN FileTemp$ FOR INPUT AS #NumFile

' Elabora il file riga per riga:
DO

   ' Ottiene una voce dal listato DIR:
   VoceDir$ = OttVoce$(NumFile, TipoVoce)

   ' Se la voce è un file:
   IF TipoVoce = TIPOFILE THEN

      ' Se la stringa SpecFile$ corrisponde, stampa la
      ' voce ed esce dal ciclo:
      IF VoceDir$ = SpecFile$ THEN
         LOCATE Riga, 1: PRINT SpecPerc$; VoceDir$;
         Riga = Riga + 1
         TipoVoce = TIPOEOF
      END IF

   ' Se la voce è una directory, esegue una chiamata
   ' ricorsiva a GuardaDir passandole la nuova directory:
   ELSEIF TipoVoce = TIPODIR THEN
      NuovoPerc$ = SpecPerc$ + VoceDir$ + "\"
      GuardaDir NuovoPerc$, Livello + 1, SpecFile$, Riga
      LOCATE 1, 1: PRINT "Ricerca in corso in "; _
         SPACE$(50);
      LOCATE 1, 21: PRINT SpecPerc$;
   END IF

LOOP UNTIL TipoVoce = TIPOEOF
```

```
    ' La ricerca nel file listato di DIR è terminata,
    ' lo chiude:
    CLOSE NumFile
END SUB

' ======================== OTTVOCE ========================
'  Questa procedura elabora le voci in un listato di DIR
'  salvato su un file.
' ==========================================================

FUNCTION OttVoce$ (NumFile, TipoVoce) STATIC

    ' Ripete il ciclo finché non legge o una voce valida
    ' o il carattere di fine file (EOF):
    DO UNTIL EOF(NumFile)
       LINE INPUT #NumFile, RigaVoce$
       IF RigaVoce$ <> "" THEN

          ' Seleziona il primo carattere della riga per la
          ' prova:
          CarProva$ = LEFT$(RigaVoce$, 1)
          IF CarProva$ <> " " AND CarProva$ <> "." _
             THEN EXIT DO
       END IF
    LOOP

    ' Ha trovato o una voce o l'EOF, e decide quale è:
    IF EOF(NumFile) THEN
       TipoVoce = TIPOEOF
       OttVoce$ = ""

    ELSE             ' Non è l'EOF, è o un file o una
                     ' directory.

       ' Compone e restituisce il nome della voce:
       NomeVoce$ = RTRIM$(LEFT$(RigaVoce$, 8))
```

```
        ' Prova se c'è un estensione da aggiungere al nome:
        EstenVoce$ = RTRIM$(MID$(RigaVoce$, 10, 3))
        IF EstenVoce$ <> "" THEN
           OttVoce$ = NomeVoce$ + "." + EstenVoce$
        ELSE
           OttVoce$ = NomeVoce$
        END IF

        ' Determina il tipo della voce, e ne restituisce il
        ' valore al punto da cui è stata chiamata OttVoce$:
        IF MID$(RigaVoce$, 15, 3) = "DIR" THEN
           TipoVoce = TIPODIR              ' Directory
        ELSE
           TipoVoce = TIPOFILE             ' File
        END IF

    END IF

END FUNCTION
```

3 I/O su file e su periferica

Il presente capitolo spiega come utilizzare le funzioni e le istruzioni di input e output (I/O) del BASIC. Tali istruzioni permettono ai programmi di accedere ai dati memorizzati nei file e di comunicare con le periferiche annesse al sistema.

Vengono illustrati inoltre processi di programmazione relativi all'estrazione, la memorizzazione e la formattazione delle informazioni. La relazione, infine, tra i file di dati e le periferiche fisiche quali schermi e tastiere, costituisce ancora materia di questo capitolo.

A lettura ultimata l'utente saprà eseguire processi quali:

- Visualizzare del testo sullo schermo
- Utilizzare in un programma un input inviato dalla tastiera
- Creare file di dati sul disco
- Memorizzare i record nei file di dati
- Leggere i record dai file di dati
- Leggere o modificare i dati nei file che non sono in formato ASCII (American Standard Code for Information Interchange)
- Comunicare con altri computer attraverso la porta seriale

3.1 Visualizzazione del testo sullo schermo

Questa sezione spiega come effettuare i seguenti processi:

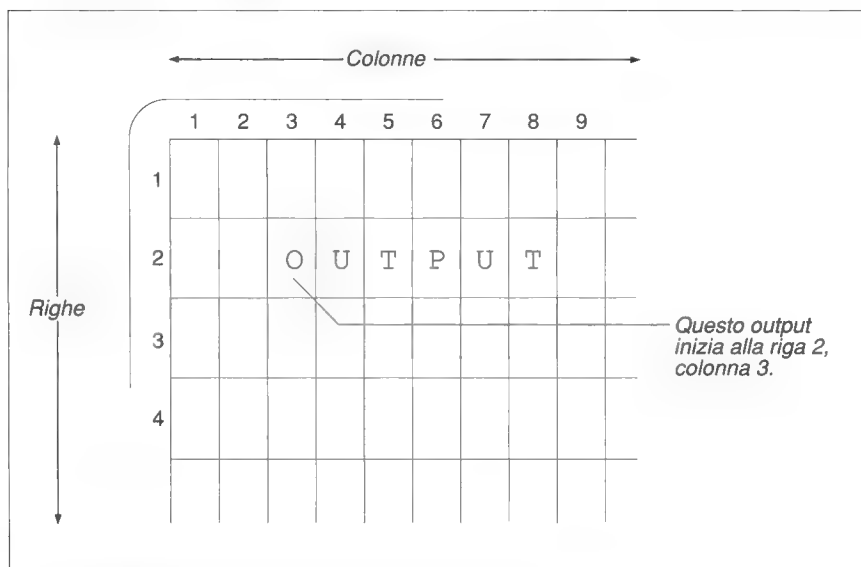
- Visualizzare del testo sullo schermo con **PRINT**
- Visualizzare sullo schermo del testo formattato con **PRINT USING**
- Eliminare con **SPC** gli spazi in una riga del testo visualizzato
- Saltare con **TAB** ad una data colonna nella riga del testo visualizzato
- Modificare con **WIDTH** i numeri delle righe o delle colonne che appaiono sullo schermo
- Aprire una finestra di testo con **VIEW PRINT**

Nota Generalmente l'output "standard" viene visualizzato sullo schermo; tuttavia è possibile reindirizzarlo dallo schermo a una periferica di uscita (quale una stampante) o a un file su disco, utilizzando i simboli > o >> dalla riga di comando del DOS. Per ulteriori informazioni relative a come reindirizzare l'output, si consulti la documentazione DOS.

3.1.1 Righe e colonne dello schermo

Le righe e le colonne per il testo visualizzato sullo schermo possono essere paragonate ad una griglia. I caratteri dell'output visualizzato sono contenuti nelle celle create dall'intersezione tra righe e colonne. Il numero standard delle colonne e delle righe dello schermo corrisponde rispettivamente a 80 e 25. Nell'illustrazione 3.1 si vede come ciascun carattere occupa una cella ben precisa della griglia.

Generalmente l'ultima riga rimane vuota, a meno che non venga attivata da una istruzione **LOCATE**. (Per ulteriori informazioni su **LOCATE**, si consulti la sezione 3.3, "Controllo del cursore di testo".)

Illustrazione 3.1 Output di testo sullo schermo

3.1.2 Visualizzazione del testo e dei numeri con PRINT

L'istruzione **PRINT** è quella più comunemente usata per visualizzare un'output sullo schermo. Con **PRINT**, è possibile visualizzare valori numerici, di stringa, o una combinazione di entrambi. Inoltre, se si utilizza **PRINT** senza argomenti viene generata una riga vuota.

Seguono alcuni commenti generali su **PRINT**:

- **PRINT** visualizza sempre i numeri seguiti da uno spazio vuoto. Se il numero è positivo viene anche preceduto da uno spazio; se è negativo, viene preceduto dal segno meno.
- L'istruzione **PRINT** può essere utilizzata per visualizzare elenchi di espressioni. Queste possono essere separate da altre espressioni mediante virgole, punti e virgola, uno o più spazi vuoti, o uno o più punti di tabulazione. Una virgola fa sì che **PRINT** salti un blocco di 14 colonne dello schermo alla successiva "zona di stampa". Un punto e virgola (o una combinazione di spazi e/o tabulazioni) tra due espressioni visualizza tali espressioni sullo schermo una di seguito all'altra senza spazi intermedi (ad eccezione degli spazi per i numeri).

continua

- Di solito, **PRINT** termina ogni riga con una sequenza di nuova riga (un ritorno a capo e un'interlinea). Tuttavia, una virgola o un punto e virgola alla fine dell'elenco delle espressioni disattiva questa predefinizione; l'output successivo comparirà sulla stessa riga, a meno che non sia troppo lungo per entrarvi.
- Se una riga supera l'ampiezza dello schermo, l'eccesso viene assorbito dalla riga successiva. Per esempio, se una riga è composta di 100 caratteri e le colonne dello schermo sono 80, i 20 caratteri in eccesso passano sulla riga successiva. Se poi la riga di 100 caratteri non inizia dall'estremo margine sinistro (p.es. se segue un'istruzione **PRINT** che termina con un punto o un punto e virgola), passano sulla riga successiva i caratteri successivi al completamento della prima.

Esempio

L'output del programma seguente mostra alcuni modi di utilizzare **PRINT**:

```
A = 2
B = -1
C = 3
X$ = "su"
Y$ = "là"
PRINT A, B, C
PRINT B, A, C
PRINT A; B; C
PRINT X$; Y$
PRINT X$, Y$;
PRINT A, B
PRINT
FOR I = 1 TO 8
   PRINT X$,
NEXT
```

Output

```
 2            -1            3
-1             2            3
 2 -1  3
sulà
su            là 2          -1
su            su            su            su            su
su            su            su
```

3.1.3 Visualizzazione dell'output formattato con PRINT USING

L'istruzione **PRINT USING** garantisce una visualizzazione più efficace dei dati rispetto a **PRINT**, specialmente dei dati numerici. Utilizzando dei caratteri speciali all'interno di una stringa, **PRINT USING** permette di specificare quante cifre di un numero (o quanti caratteri di una stringa) sono da visualizzare, e se vanno precedute da un segno più (+) o un segno di valuta ($ o Lit.), e così via.

Esempio

L'esempio che segue mostra come può essere utilizzato **PRINT USING**. Analogamente a **PRINT**, le espressioni dell'elenco possono essere separate le une dalle altre mediante virgole, punti e virgola, o tabulazioni.

```
X = 441.2318

PRINT USING "Il numero con 3 cifre decimali ###.###";X
PRINT USING "Il numero con un segno di dollaro $$##.##";X
PRINT USING "Il numero con un altro simbolo di valuta Lit. _
   ###";X
PRINT USING "Il numero in formato esponenziale #.###^^^^";X
PRINT USING "Numeri con segni più +###"  ; X; 99.9
```

Output

```
Il numero con tre cifre decimali 441.232
Il numero con un segno di dollaro $441.23
Il numero con un altro simbolo di valuta Lit. 441
Il numero in formato esponenziale 0.441E+03
Numeri con segni più +441  Numeri con segni più +100
```

Per maggiori informazioni su **PRINT USING**, si consulti il Consulente QB.

3.1.4 Salto di spazi ed avanzamento ad una colonna specifica

Utilizzando l'istruzione **SPC** (*n*) in un'istruzione **PRINT**, è possibile saltare *n* spazi in una riga dell'output visualizzato, come indica l'output dell'esempio successivo:

```
PRINT "         1         2         3"
PRINT "123456789012345678901234567890"
PRINT "Nome"; SPC(10); "Cognome"
```

Output

```
         1         2         3
123456789012345678901234567890
Nome          Cognome
```

Utilizzando l'istruzione **TAB** (*n*) in un'istruzione **PRINT**, è possibile saltare alla *n* esima colonna (a partire dal lato sinistro dello schermo) in una riga dell'output visualizzato. L'esempio seguente utilizza **TAB** per produrre l'output uguale a quello sopra indicato:

```
PRINT "         1         2         3"
PRINT "123456789012345678901234567890"
PRINT "Nome";TAB(15); "Cognome"
```

Né **SPC** né **TAB** possono essere utilizzati da soli per posizionare l'output visualizzato sullo schermo; essi devono essere utilizzati solo nelle istruzioni **PRINT**.

3.1.5 Modifica del numero delle colonne o delle righe

Mediante l'istruzione **WIDTH** *colonne*, è possibile controllare il numero massimo di caratteri che appaiono in un'unica riga di output. Tale istruzione in realtà modifica la dimensione dei caratteri visualizzati sullo schermo, così che un numero maggiore o minore di caratteri possa entrare in una riga.

Ad esempio, **WIDTH 40** rende i caratteri più larghi, in modo che una riga può contenere al massimo 40 caratteri. **WIDTH 80** restringe i caratteri, così la riga ne può contenere 80. I numeri 40 e 80 sono gli unici valori validi per l'argomento *colonne*.

Nei computer dotati di scheda video EGA o VGA, l'istruzione **WIDTH** è inoltre in grado di controllare il numero di righe che appaiono sullo schermo in questo modo:

WIDTH [*colonne*] [, *righe*]

Il numero delle *righe* può essere 25, 30, 43, 50 o 60, a seconda del tipo di adattatore schermo in uso e della modalità schermo impostata con una precedente istruzione **SCREEN**.

3.1.6 Creazione di una finestra di testo

Finora, l'output è stato visualizzato su tutto lo schermo. Tuttavia, con l'istruzione **VIEW PRINT**, è possibile concentrarlo orizzontalmente in una "finestra di testo", cioè solo in una parte dello schermo. La sintassi dell'istruzione **VIEW PRINT** è:

VIEW PRINT [*primariga* **TO** *ultimariga*]

I valori di *primariga* e *ultimariga* specificano l'inizio e la fine della finestra.

Una finestra di testo permette inoltre di controllare lo scorrimento del testo sullo schermo. Senza la finestra di testo, una volta che il testo raggiunge il bordo inferiore dello schermo, fa scorrere via l'output grafico o di testo che si trovasse nella parte superiore. Dopo un'istruzione **VIEW PRINT**, invece, scorrono solo le righe comprese tra la parte superiore ed inferiore della finestra. Ciò significa che è possibile etichettare l'output visualizzato sullo schermo in alto e/o in basso, senza doversi preoccupare che le etichette scorrano via se appaiono troppe righe di dati. Si può inoltre utilizzare l'istruzione **CLS 2** per cancellare solo la finestra di testo, lasciando intatto il contenuto del resto dello schermo. Sulla creazione di finestre per output grafici, si consulti la sezione 5.5, "Definizione di una finestra grafica".

Esempio

Gli effetti di una istruzione **VIEW PRINT** si possono vedere esaminando l'output dell'esempio successivo:

```
CLS
LOCATE 3, 1
PRINT "Al di sopra della finestra di testo; non scorre."

LOCATE 4, 1
PRINT STRING$(60, "_")         ' Stampa linee orizzontali
                               ' sopra e
LOCATE 11, 1
PRINT STRING$(60, "_")         ' sotto la finestra di testo.

PRINT "Al di sotto della finestra di testo; non scorre."

VIEW PRINT 5 TO 10             ' La finestra di testo va
                               ' dalla riga 5 alla 10.

FOR I = 1 TO 20                ' Stampa numeri e testo
PRINT I; "una riga di testo"   ' nella finestra.
NEXT

DO: LOOP WHILE INKEY$ = ""     ' Aspetta una pressione
                               ' di tasto.
CLS 2                          ' Cancella la finestra
                               ' solamente.
END
```

Output (prima di premere un tasto)

```
Al di sopra della finestra di testo: non scorre.
________________________________________________
 16 una riga di testo
 17 una riga di testo
 18 una riga di testo
 19 una riga di testo
 20 una riga di testo

________________________________________________
Al di sotto della finestra di testo; non scorre.
```

Output (dopo aver premuto un tasto)

3.2 Input dalla tastiera

Questa sezione spiega come abilitare programmi in BASIC ad accettare l'input inviato dalla tastiera:

- **INPUT**
- **LINE INPUT**
- **INPUT$**
- **INKEY$**

Nota Generalmente l'input standard è quello digitato sulla tastiera. Tuttavia, si può utilizzare il simbolo < del DOS per specificare l'input standard da un file o da un'altra periferica di entrata invece che dalla tastiera. (Per maggiori informazioni relative a come reindirizzare l'input, consultare la documentazione DOS.)

3.2.1 L'istruzione INPUT

L'istruzione **INPUT** riceve l'informazione digitata dall'utente e la memorizza in un elenco di variabili, come si vede qui:

```
INPUT A%, B, C$
INPUT D$
PRINT A%, B, C$, D$
```

Output

```
? 6.6,45,una stringa
? "due, tre"
 7            45           una stringa   due, tre
```

L'istruzione **INPUT** merita alcuni commenti:

- Un'istruzione **INPUT** sollecita l'utente con un punto interrogativo (?) seguito da un cursore lampeggiante.
- L'istruzione **INPUT** è seguita da uno o più nomi di variabili. Se esistono più variabili, queste sono separate da virgole.
- Il numero delle costanti digitate dall'utente in seguito al sollecito di **INPUT** deve essere uguale al numero di variabili comprese nella stessa istruzione **INPUT**.

continua

- I valori che l'utente digita devono essere dello stesso tipo delle variabili dell'elenco che segue l'istruzione **INPUT**. In altri termini, si deve digitare un numero se la variabile è di tipo intero, intero lungo, in precisione semplice o in precisione doppia; una stringa se la variabile è di tipo stringa.
- Poiché le costanti in un'elenco di input devono essere separate da virgole, un input di una costante a stringa deve essere racchiuso tra doppie virgolette. Le doppie virgolette garantiscono che la stringa venga considerata come un'unità e non scomposta in due o più parti.

Se si infrange una delle ultime tre condizioni, il BASIC visualizza il messaggio di errore `Ricominciare da capo`. Questo messaggio compare finché l'input non concorda per numero e tipo con la lista delle variabili.

Se da un sollecito per l'input si desiderano maggiori informazioni che il semplice punto di domanda, far visualizzare del testo nel modo seguente:

```
INPUT "Qual è l'ora esatta (ore, minuti)"; Ora$, Min$
```

Viene visualizzata la seguente richiesta:

```
Qual è l'ora esatta (ore, minuti)?
```

Si noti il punto e virgola tra il sollecito e le variabili dell'input. Il punto e virgola fa in modo che il punto interrogativo sia incluso nella richiesta. Se si vuole eliminare il punto interrogativo, inserire una virgola tra la richiesta e l'elenco delle variabili:

```
INPUT "Digitare l'ora esatta (ore, minuti): ", Ora$, Min$
```

Viene visualizzata la seguente richiesta:

```
Digitare l'ora esatta (ore, minuti):
```

3.2.2 L'istruzione LINE INPUT

Con l'istruzione **LINE INPUT** il programma accetta un'intera riga di testo contenente anche virgole e spazi, senza che l'input debba essere racchiuso tra doppie virgolette. L'istruzione **LINE INPUT**, come dice il nome, legge una riga di input (terminata dalla pressione del tasto INVIO) dalla tastiera e la memorizza in una singola variabile a stringa. Diversamente dall'istruzione **INPUT**, l'istruzione **LINE INPUT** non visualizza come predefinizione un punto di domanda quando richiede l'input; permette, tuttavia, di visualizzare una stringa di sollecito.

L'esempio seguente mostra la differenza tra **INPUT** e **LINE INPUT**:

```
' Assegna l'input a tre variabili diverse:
INPUT "Digitare tre valori separati da virgole: _
   ", A$, B$, C$

' Assegna l'input a una sola variabile (le virgole non
' vengono considerate delimitatorie):
LINE INPUT "Digitare gli stessi tre valori: ", D$

PRINT "A$ = "; A$
PRINT "B$ = "; B$
PRINT "C$ = "; C$
PRINT "D$ = "; D$
```

Output

```
Digitare tre valori separati da virgole: Tizio, _
Caio, e Sempronio
Digitare gli stessi tre valori: Tizio, Caio, _
e Sempronio
A$ = Tizio
B$ = Caio
C$ = e Sempronio
D$ = Tizio, Caio, e Sempronio
```

Sia con **INPUT** che con **LINE INPUT**, l'input termina quando l'utente preme il tasto INVIO, che fa inoltre avanzare il cursore alla riga seguente. Come mostra l'esempio seguente, un punto e virgola tra la parola chiave **INPUT** e la stringa del sollecito fa in modo che il cursore rimanga sulla stessa riga:

```
INPUT "Primo valore: ", A
INPUT; "Secondo valore: ", B
INPUT "    Terzo valore: ", C
```

Ciò che segue mostra alcuni esempi di input al programma precedente e la posizione dei prompt:

```
Primo valore: 5
Secondo valore: 4     Terzo valore: 3
```

3.2.3 La funzione INPUT$

Sia **INPUT** che **LINE INPUT** aspettano che venga premuto il tasto INVIO prima di memorizzare ciò che è stato digitato; cioè, essi leggono una riga dell'input, quindi la assegnano alle variabili del programma. Al contrario, la funzione **INPUT$** (*numero*) non aspetta che venga premuto il tasto INVIO, ma legge direttamente un determinato numero di caratteri. Ad esempio, la riga seguente in un programma legge tre caratteri digitati dall'utente, quindi memorizza la stringa di tre caratteri nella variabile `Test$`:

```
Test$ = INPUT$(3)
```

Diversamente dall'istruzione **INPUT**, la funzione **INPUT$** non sollecita dati dall'utente, né invia i caratteri dell'input sullo schermo. Inoltre, poiché **INPUT$** è una funzione, essa da sola non costituisce un'istruzione completa. **INPUT$** deve apparire all'interno di un'espressione, come nel seguente esempio:

```
INPUT X                      ' INPUT è un'istruzione.
PRINT INPUT$(1)              ' INPUT$ è una funzione, e deve
Y$ = INPUT$(1)               ' far parte di un'espressione.
```

La funzione **INPUT$** legge l'input dalla tastiera come un flusso di caratteri non formattati. Diversamente da **INPUT** o da **LINE INPUT**, **INPUT$** legge qualunque tasto premuto, inclusi i tasti di controllo come ESC o BACKSPACE. Ad esempio, premendo il tasto INVIO per cinque volte, si assegnano cinque caratteri di invio alla variabile `Test$` nella riga successiva:

```
Test$ = INPUT$(5)
```

3.2.4 La funzione INKEY$

La funzione **INKEY$** completa l'elenco delle funzioni e istruzioni BASIC che leggono un input dalla tastiera. Quando il BASIC incontra un'espressione contenente la funzione **INKEY$**, esso controlla se l'utente ha premuto un tasto a partire:

- Dall'ultima volta che esso ha rilevato la presenza di un'espressione **INKEY$**
- Dall'inizio del programma, se questa è la prima volta che compare **INKEY$**

Se non è stato premuto alcun tasto dall'ultima volta che il programma ha controllato, **INKEY$** restituisce una stringa nulla (`""`). Se invece è stato premuto un tasto, **INKEY$** restituisce il carattere che corrisponde a quel tasto.

Esempio

La differenza più importante tra **INKEY$** e le altre istruzioni e funzioni trattate in questa sezione, è che **INKEY$** fa sì che il programma continui ad operare mentre controlla l'input. Al contrario, **LINE INPUT**, **INPUT$** e **INPUT** sospendono l'esecuzione del programma finché non appare un input, come mostra questo esempio:

```
PRINT "Premere un tasto per iniziare. ";
PRINT "Premere di nuovo per uscire."

' Non fa nulla finché l'utente non preme un tasto:
Inizio$ = INPUT$(1)

I = 1

' Stampa i numeri da uno a un milione. Durante l'operazione
' controlla se è stato premuto un tasto:
DO
PRINT I
I = I + 1

' Continua a iterare finché I non è maggiore di un milione
' o finché non viene premuto un tasto:
LOOP UNTIL I > 1000000 OR INKEY$ <> ""
```

3.3 Controllo del cursore di testo

Quando il testo stampato viene visualizzato sullo schermo, il cursore del testo indica il punto in cui comparirà l'output del programma, o l'input digitato dall'utente. Nell'esempio seguente, dopo che l'istruzione **INPUT** visualizza il sollecito di 6 caratteri `Nome: `, il cursore aspetta l'input alla riga 1, colonna 7:

```
' Cancella lo schermo; inizia la stampa alla riga 1,
' colonna 1:
CLS
INPUT "Nome: ", Nome$
```

Nell'esempio successivo, il punto e virgola alla fine della seconda istruzione **PRINT** lascia il cursore alla riga 2, colonna 33:

```
CLS
PRINT

' Trentadue caratteri nella seguente riga:
PRINT "Premere un tasto per continuare.";
PRINT INPUT$(1)
```

Le sezioni 3.3.1–3.3.3 mostrano come controllare la posizione del cursore del testo, come modificarne la forma e come ottenere informazioni circa la sua posizione.

3.3.1 Posizionamento del cursore

Le istruzioni e le funzioni di input e output finora trattate non consentono di controllare esattamente il punto in cui verrà visualizzato l'output o in cui viene posizionato il cursore dopo la visualizzazione dell'output. Le richieste di input e l'output cominciano sempre sulla sinistra dello schermo e scorrono una riga alla volta, dall'alto in basso, a meno che non si utilizzi un punto e virgola nelle istruzioni **PRINT** e **INPUT** per disattivare il ritorno a capo del cursore.

Le istruzioni **SPC** e **TAB**, di cui si è parlato nella sezione 3.1.4, "Salto di spazi ed avanzamento ad una colonna specifica", offrono un certo controllo sulla posizione del cursore, permettendo di portarlo su qualsiasi colonna all'interno di una determinata riga.

L'istruzione **LOCATE** estende questo controllo. La sintassi per **LOCATE** è

LOCATE [*riga*] [, [*colonna*] [, [*cursore*] [, [*inizio*] [, *fine*]]]]

L'utilizzo dell'istruzione **LOCATE** permette di posizionare il cursore su qualsiasi *riga* e *colonna* dello schermo, come mostra l'output nell'esempio seguente:

```
CLS
FOR Riga = 9 TO 1 STEP -2
   Colonna = 2 * Riga
   LOCATE Riga, Colonna
   PRINT "12345678901234567890";
NEXT
```

Output

```
1234567890123456789O
    1234567890123456789O
        1234567890123456789O
            1234567890123456789O
                1234567890123456789O
```

3.3.2 Modifica della forma del cursore

Gli argomenti opzionali *cursore*, *inizio* e *fine*, esposti nella sintassi dell'istruzione **LOCATE**, permettono inoltre di modificare la forma del cursore e di renderlo visibile o invisibile. Il valore 1 dell'argomento *cursore* rende il cursore visibile, mentre il valore 0 lo rende invisibile. Gli argomenti *inizio* e *fine* controllano l'altezza del cursore, se questo è attivato, specificandone l'uno l'inizio e l'altro la fine delle righe di "pixel". (Ciascun carattere sullo schermo è composto da righe di pixel, che sono punti luminosi sullo schermo.) Se il cursore è alto come una riga di testo, allora la riga di pixel superiore del cursore ha il valore 0, mentre quella inferiore ha il valore 7 o 13, a seconda del tipo di adattatore di schermo che si possiede. (Per quello monocromatico il valore è 13; per quello a colori è 7.)

E' possibile accendere il cursore o modificarne la forma senza cambiarne la posizione. Ad esempio, l'istruzione seguente lascia il cursore alla posizione occupata dopo l'istruzione **PRINT** o **INPUT**, quindi ne riduce l'altezza alla metà di un carattere:

```
LOCATE , , 1, 2, 5       ' Gli argomenti della riga e colonna
                         ' sono entrambi opzionali.
```

I seguenti esempi mostrano diverse forme che il cursore può assumere utilizzando diversi valori di *inizio* e *fine* su di un monitor a colori. Ciascuna istruzione **LOCATE**, nella colonna a sinistra, è seguita dall'istruzione:

```
INPUT "Prompt: ", X$
```

Istruzione	*Prompt di input e forma del cursore*
`LOCATE , , 1, 0, 7`	Prompt: █
`LOCATE , , 1, 0, 3`	Prompt: ▀
`LOCATE , , 1, 4, 7`	Prompt: ▄
`LOCATE , , 1, 6, 2`	Prompt: ▀▄

Negli esempi precedenti si noti che rendendo l'argomento *inizio* maggiore dell'argomento *fine* risulta che il cursore viene diviso in due.

3.3.3 Richiesta di informazioni sulla posizione del cursore

Le funzioni **CSRLIN** e **POS** (*n*) sono l'inverso dell'istruzione **LOCATE**; mentre con **LOCATE** il programma assegna una posizione al cursore, **CSRLIN** e **POS** (*n*) informano il programma sulla posizione del cursore. La funzione **CSRLIN** restituisce la riga, la funzione **POS** (*n*) la colonna sulla quale si trova il cursore.

L'argomento *n* per **POS** (*n*) è noto come argomento "fittizio"; cioè, *n* è un parametro formale per una qualsiasi espressione numerica. Ad esempio, **POS(0)** e **POS(1)** restituiscono entrambi lo stesso valore.

Esempio

L'esempio seguente utilizza la funzione **POS** (*n*) per visualizzare 50 asterischi in righe di 13 asterischi l'una:

```
FOR I% = 1 TO 50
   PRINT "*";                   ' Stampa un asterisco
                                ' lasciando il cursore sulla
                                ' stessa riga.
   IF POS(1) > 13 THEN PRINT    ' Se il cursore ha superato
                                ' la colonna 13, avanza alla
                                ' riga successiva.
NEXT
```

Output

```
*************
*************
*************
***********
```

3.4 Utilizzo dei file di dati

I file di dati sono degli archivi nei quali le informazioni rimangono memorizzate in permanenza. Nei programmi BASIC, essi semplificano notevolmente i seguenti processi:

- Creare, manipolare e memorizzare grandi quantità di dati
- Accedere a diversi insiemi di dati con un solo programma
- Utilizzare lo stesso insieme di dati in diversi programmi

Le sezioni successive introducono i concetti di record e campo e confrontano i diversi modi di accesso ai file di dati del BASIC. Dopo aver letto le sezioni 3.4.1–3.4.7, sarà possibile eseguire le seguenti operazioni:

- Creare dei nuovi file di dati
- Aprire dei file già esistenti e leggere il loro contenuto
- Aggiungere nuove informazioni ad un file di dati già esistente
- Modificare il contenuto di un file di dati già esistente

3.4.1 Organizzazione dei file di dati

Un file di dati contiene una serie di informazioni raggruppate in "record". Ogni record è ulteriormente suddiviso in "campi", o voci, ricorrenti all'interno di ciascun record. Azzardando un paragone tra un file di dati ed un archivio contenente domande di impiego, un record equivale ad una di queste domande. Un campo è invece paragonabile ad una singola voce che si trova in ogni domanda, come ad esempio il cognome.

Nota Se non si vuole accedere ad un file utilizzandone i record, ma si preferisce considerarlo come una sequenza di byte non formattati, consultare la sezione 3.4.7, "I/O su file binari".

3.4.2 File ad accesso sequenziale e ad accesso casuale

I termini "file ad accesso sequenziale" e "file ad accesso casuale" si riferiscono a due modi differenti di memorizzare e di accedere ai dati su un disco dall'interno di un programma BASIC. Nei file ad accesso sequenziale, i dati sono organizzati sequenzialmente e non è possibile accedere direttamente ad un record specifico senza prima passare per i record intermedi. Al contrario, nei file ad accesso casuale si può richiamare qualsiasi record semplicemente specificandone il numero, accelerando i tempi di ricerca. Questo accade in maniera analoga alla ricerca di una canzone registrata su di un nastro (accesso sequenziale: bisogna far scorrere il nastro fino a trovare la canzone desiderata), o incisa su un disco (accesso casuale: basta abbassare la puntina al punto in cui inizia la canzone desiderata).

Nota Sebbene non ci sia modo di accedere direttamente ad un record specifico in un file ad accesso sequenziale, l'istruzione **SEEK** consente di saltare direttamente ad un byte specifico del file. Per maggiori informazioni su questa istruzione, si consulti la sezione 3.4.7, "I/O su file binari ".

3.4.3 Apertura di un file di dati

Prima che il programma possa leggere, modificare o aggiungere dati in un file, è necessario innanzi tutto aprire il file con l'istruzione **OPEN**, che può essere utilizzata anche per creare un nuovo file. L'elenco che segue descrive le varie funzioni dell'istruzione **OPEN**:

- Creare un nuovo file di dati e aprirlo in modo da potervi aggiungere dei record:

  ```
  ' Nella directory corrente non si trova un file di nome
  ' PREZZI.DAT:
  OPEN "PREZZI.DAT" FOR OUTPUT AS #1
  ```

- Aprire un file di dati già esistente e sovrascrivervi nuovi record, cancellando i dati preesistenti:

  ```
  ' Un file di nome PREZZI.DAT si trova già nella directory
  ' corrente; vi si possono registrare nuovi dati, ma
  ' quelli preesistenti verranno cancellati:
  OPEN "PREZZI.DAT" FOR OUTPUT AS #1
  ```

- Aprire un file di dati già esistente ed aggiungervi nuovi record alla fine, conservandone i dati registrati in precedenza:

  ```
  OPEN "PREZZI.DAT" FOR APPEND AS #1
  ```

  La modalità **APPEND** creerà un nuovo file se nella directory corrente non esiste già un file con il dato nome.

- Aprire un file di dati già esistente per leggervi i record:

  ```
  OPEN "PREZZI.DAT" FOR INPUT AS #1
  ```

  Per maggiori informazioni circa le modalità **INPUT**, **OUTPUT** e **APPEND**, si consulti la sezione 3.4.5, "Utilizzo di file ad accesso sequenziale".

- Aprire un file di dati già esistente (o crearne uno nuovo se non esiste un file con quel nome), quindi leggervi o scrivervi dei record a lunghezza fissa:

  ```
  OPEN "PREZZI.DAT" FOR RANDOM AS #1
  ```

  Per ulteriori informazioni circa questa modalità, si consulti la sezione 3.4.6, "Utilizzo di file ad accesso casuale".

- Aprire un file di dati già esistente (o crearne uno nuovo se non esiste un file con quel nome), quindi leggere dati o aggiungerne nuovi a partire da una qualsiasi posizione di byte nel file:

  ```
  OPEN "PREZZI.DAT" FOR BINARY AS #1
  ```

  Per ulteriori informazioni circa questa modalità, si consulti la sezione 3.4.7, "I/O su file binari".

3.4.3.1 Numeri dei file in BASIC

L'istruzione **OPEN** non solo determina una modalità per l'I/O dei dati a un file (**OUTPUT**, **INPUT**, **APPEND**, **RANDOM** o **BINARY**), ma assegna al file un numero specifico. Questo numero, compreso tra 1 e 255, viene poi utilizzato dalle istruzioni per l'I/O del programma per riferire al file. Finché il file rimane aperto, il numero resta associato al file: quando il file viene chiuso, esso può essere riassegnato ad un altro file. (Per ulteriori informazioni su come chiudere i file, si consulti la sezione 3.4.4.)
I programmi BASIC possono aprire più di un file alla volta.

La funzione **FREEFILE** serve per trovare un numero di file non utilizzato. Questa funzione restituisce il primo numero disponibile che può essere associato con un file in un'istruzione **OPEN**. Per esempio, **FREEFILE** potrebbe restituire il valore 3 dopo le seguenti istruzioni **OPEN**:

```
OPEN "TEST1.DAT" FOR RANDOM AS #1
OPEN "TEST2.DAT" FOR RANDOM AS #2
NumFile = FREEFILE
OPEN "TEST3.DAT" FOR RANDOM AS #NumFile
```

La funzione **FREEFILE** risulta particolarmente utile per creare delle procedure di libreria che aprono i file. Con **FREEFILE** non è necessario passare a queste procedure informazioni relative al numero dei file aperti.

3.4.3.2 Nomi dei file in BASIC

Nelle istruzioni **OPEN**, i nomi dei file possono corrispondere ad una qualsiasi espressione a stringa, composta da una qualsiasi combinazione dei seguenti caratteri:

- Lettere maiuscole o minuscole (A–Z e a–z) non accentate
- Cifre da 0 a 9
- I seguenti caratteri speciali:

  ( ) { } @ # $ % ^ & ! - _ ' ~

L'espressione a stringa inoltre può contenere opzionalmente una specifica dell'unità disco, e una specifica completa o parziale del percorso di ricerca. Ciò significa che un programma BASIC può utilizzare file di dati in un'altra unità disco o in una directory diversa da quella nella quale il programma è in esecuzione. Ad esempio, le seguenti istruzioni **OPEN** sono tutte valide:

```
OPEN "..\VOTI.TRI" FOR INPUT AS #1

OPEN "a:\STIPENDI\1988.MAN" FOR INPUT AS #2

NomeFile$ = "FILETEMP"
OPEN NomeFile$ FOR OUTPUT AS #3

NomeBase$ = "INVENT"
OPEN NomeBase$ + ".DAT" FOR OUTPUT AS #4
```

Anche il DOS impone delle restrizioni ai nomi di file: non si possono utilizzare più di otto caratteri per il nome di base (quello alla sinistra del punto facoltativo) e non più di tre caratteri per l'estensione (alla destra del punto). Nei programmi BASIC, i nomi di file troppo lunghi vengono troncati nel seguente modo:

Nome del file nel programma	*Nome del file risultante in DOS*
`Prog@Data@File`	`PROG@DAT.A@F` Poiché il nome supera gli 11 caratteri, il BASIC considera le prime 8 lettere il nome di base, inserisce un punto (.) ed utilizza le successive 3 lettere come estensione. Tutto il resto viene scartato.
`Posta#.Versione1`	`POSTA#.VER` Poiché il nome di base (Posta#) ha meno di 8 caratteri, e l'estensione (Versione1) ne ha più di 3, questa si riduce a tre caratteri.

Nome del file nel programma	*Nome del file risultante in DOS*
`Analisi_Appunti.Bak`	Causa il messaggio di errore `Nome di file non valido`. Se al nome di base si ha intenzione di aggiungere un'estensione esplicita (in questo caso `.BAK`), esso non deve superare gli 8 caratteri.

Il DOS non distingue tra maiuscole e minuscole; tutte le lettere minuscole nei nomi dei file vengono convertite in lettere maiuscole. Pertanto, non si deve fare affidamento sulla combinazione di caratteri maiuscoli o minuscoli per distinguere i file. Ad esempio, se un file sul disco è già stato chiamato INVEST.DAT, l'istruzione **OPEN** seguente sovrascriverebbe quel file, distruggendone tutte le informazioni già memorizzate:

```
OPEN "Invest.Dat" FOR OUTPUT AS #1
```

3.4.4 Chiusura di un file di dati

La chiusura di un file di dati ha due conseguenze importanti: innanzi tutto, qualsiasi dato che si trova ancora nel buffer del file (un'area di memoria di transito) viene scritto sul file; in secondo luogo, il numero di file ad esso associato viene liberato e può essere utilizzato da un'altra istruzione **OPEN**.

L'istruzione **CLOSE** viene utilizzata per chiudere un file dall'interno del programma. Ad esempio, se il file PREZZI.DAT è stato aperto con l'istruzione:

```
OPEN "PREZZI.DAT" FOR OUTPUT AS #1
```

allora l'istruzione `CLOSE #1` fa terminare l'output indirizzato a PREZZI.DAT. Se PREZZI.DAT è stato aperto con l'istruzione:

```
OPEN "PREZZI.DAT" FOR OUTPUT AS #2
```

allora l'istruzione giusta per far terminare l'output è `CLOSE #2`. Un'istruzione **CLOSE** senza gli argomenti del numero del file chiude tutti i file aperti.

Un file di dati viene chiuso anche quando si verifica una di queste due condizioni:

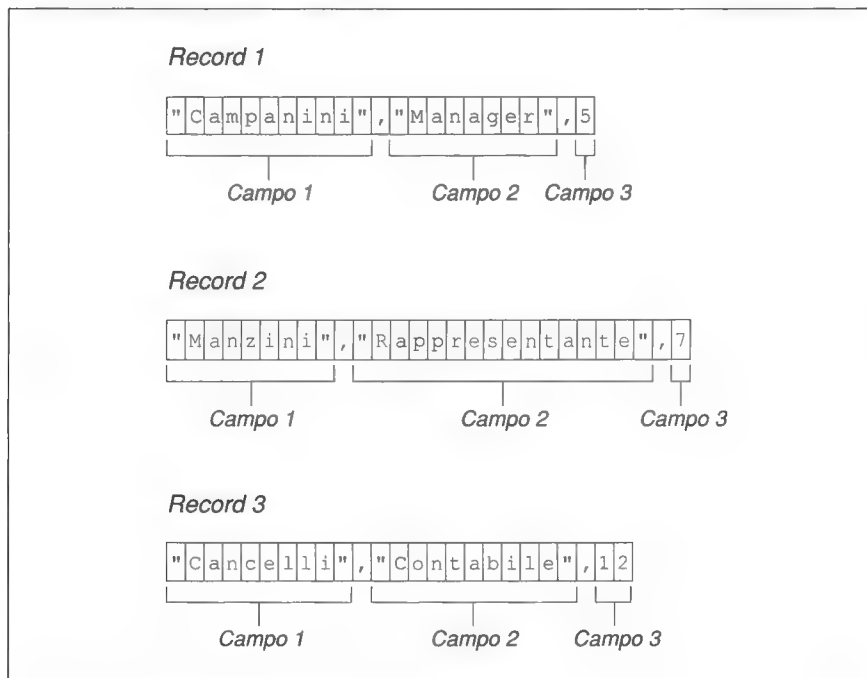
- Quando il programma BASIC che sta eseguendo le operazioni di I/O termina l'esecuzione (la terminazione del programma determina sempre la chiusura di tutti i file di dati aperti).
- Quando il programma che sta eseguendo le operazioni di I/O trasferisce il controllo ad un altro programma per mezzo dell'istruzione **RUN**.

3.4.5 Utilizzo di file ad accesso sequenziale

Questa sezione spiega come vengono organizzati i record nei file di dati ad accesso sequenziale e, successivamente, descrive il modo per leggere o scrivere dati in un file ad accesso sequenziale aperto.

3.4.5.1 Record nei file ad accesso sequenziale

Poiché i file ad accesso sequenziale sono file ASCII (di testo), è possibile utilizzare qualsiasi elaboratore di testo per visualizzarli o modificarli. In un file ad accesso sequenziale i record vengono memorizzati come una singola riga di testo, chiusa con una sequenza a capo-interlinea (CR-LF). Ciascun record è suddiviso in campi, o gruppi ripetuti di dati che appaiono nello stesso ordine in ogni record. L'illustrazione 3.2 illustra il modo in cui potrebbero apparire tre record in un file ad accesso sequenziale.

Illustrazione 3.2 Record in file ad accesso sequenziale

Si noti che ciascun record in un file ad accesso sequenziale può avere una lunghezza diversa, così come i campi in record diversi.

Il tipo della variabile in cui si memorizza un campo determina dove esso inizia e termina. (Per degli esempi di lettura e di memorizzazione di campi da record, si consultino le sezioni 3.4.5.2 e 3.4.5.6.) Ad esempio, se il programma legge un campo con una variabile a stringa, la fine di quel campo può essere determinata in uno dei seguenti modi:

- Con doppie virgolette (") se la stringa inizia con doppie virgolette
- Con una virgola (,) se la stringa non inizia con doppie virgolette
- Con un CR-LF se il campo si trova alla fine del record

D'altra parte, se il programma legge un campo con una variabile numerica, allora la fine di quel campo può essere determinata in uno dei seguenti modi:

- Con una virgola
- Con uno o più spazi
- Con un CR-LF

3.4.5.2 Inserimento di dati in un file ad accesso sequenziale

E' possibile aggiungere dati ad un nuovo file ad accesso sequenziale subito dopo averlo aperto per la scrittura di record con un'istruzione **OPEN** *nomefile* **FOR OUTPUT**. Per scrivere i record nel file, viene utilizzata l'istruzione **WRITE #**.

Un file ad accesso sequenziale può essere aperto per la lettura o per la scrittura, ma non per entrambe contemporaneamente. Durante la scrittura su di un file ad accesso sequenziale, per rileggere i dati memorizzati si deve prima chiudere il file e poi riaprirlo per l'input.

Esempio

Il breve programma seguente crea un file ad accesso sequenziale chiamato PREZZI.DAT, ed aggiunge al file i dati inviati dalla tastiera. In questo programma, l'istruzione **OPEN** crea il file e lo dispone a ricevere i record; successivamente, **WRITE #** scrive ciascun record nel file. Si noti che il numero utilizzato nell'istruzione **WRITE #** è lo stesso assegnato al nome del file PREZZI.DAT nell'istruzione **OPEN**.

```
' Crea un file detto PREZZI.DAT e lo apre per
' la scrittura di nuovi dati:

OPEN "PREZZI.DAT" FOR OUTPUT AS #1
```

```
DO
   ' Continua a inserire record in PREZZI.DAT finché
   ' l'utente non preme <INVIO> senza aver digitato il
   ' nome di una ditta:
   INPUT "Ditta (premere <INVIO> per uscire): ", Ditta$

   IF Ditta$ <> "" THEN

      ' Legge gli altri campi del record:
      INPUT "Stile: ", Stile$
      INPUT "Taglia: ", Taglia$
      INPUT "Tinta: ", Tinta$
      INPUT "Quantità: ", Quant

      ' Inserisce il nuovo record nel file con
      ' l'istruzione WRITE #:
      WRITE #1, Ditta$, Stile$, Taglia$, Tinta$, Quant
   END IF
LOOP UNTIL Ditta$ = ""

' Chiude PREZZI.DAT (terminando l'output al file):
CLOSE #1
END
```

Attenzione Se, nell'esempio precedente, si possedeva già un file PREZZI.DAT sul disco, la modalità **OUTPUT** data nell'istruzione **OPEN** cancellerebbe il contenuto di PREZZI.DAT prima di scrivervi i nuovi dati. Volendo aggiungere nuovi dati alla fine di un file esistente, senza cancellare quelli già inseriti, si utilizzi la modalità **APPEND** dell'istruzione **OPEN**. Per ulteriori informazioni riguardo a tale modalità, si consulti la sezione 3.4.5.4, "Aggiunta di dati in un file ad accesso sequenziale".

3.4.5.3 Lettura di dati da un file ad accesso sequenziale

I dati di un file ad accesso sequenziale possono essere letti dopo aver aperto il file con l'istruzione **OPEN** *nomefile* **FOR INPUT**. Per leggere i dati di un file un campo per volta viene utilizzata l'istruzione **INPUT #**. (Per ulteriori informazioni relative ad altre istruzioni e funzioni per l'input da file utilizzabili con file ad accesso sequenziale, si consulti la sezione 3.4.5.6 "Alternative per la lettura di dati da un file ad accesso sequenziale".)

Esempio

L'esempio successivo apre il file di dati PREZZI.DAT creato nell'esempio precedente, e ne legge i record, visualizzando sullo schermo l'intero record se il valore dell'elemento `Quant` è minore del valore digitato.

L'istruzione **INPUT #** legge un record alla volta da PREZZI.DAT, assegnandone i campi alle variabili `Ditta$`, `Stile$`, `Taglia$`, `Tinta$`, e `Quant`. Poiché si tratta di un file ad accesso sequenziale, i record vengono letti nell'ordine di inserimento (dal primo all'ultimo digitato).

La funzione **EOF** (fine del file) controlla se **INPUT #** ha già letto l'ultimo record. In caso affermativo, **EOF** restituisce il valore −1 (vero) ed il ciclo di lettura dei dati termina; in caso contrario, **EOF** restituisce il valore 0 (falso) ed il record successivo viene letto dal file.

```
OPEN "PREZZI.DAT" FOR INPUT AS #1

INPUT "Visualizzare i pezzi il cui quantitativo è _
   minore di"; Riordina

DO UNTIL EOF(1)
   INPUT #1, Ditta$, Stile$, Taglia$, Tinta$, Quant
   IF Quant < Riordina THEN
      PRINT Ditta$, Stile$, Taglia$, Tinta$, Quant
   END IF
LOOP

CLOSE #1
END
```

3.4.5.4 Aggiunta di dati in un file ad accesso sequenziale

Come è stato detto precedentemente, per aggiungere ulteriori dati alla fine di un file ad accesso sequenziale sul disco, non si può semplicemente aprire il file nella modalità **OUTPUT** e iniziare a scrivervi i dati. Infatti, non appena un file ad accesso sequenziale viene aperto nella modalità **OUTPUT**, ne viene distrutto il contenuto. E' necessario invece utilizzare la modalità **APPEND**, come mostrato nell'esempio seguente:

```
OPEN "PREZZI.DAT" FOR APPEND AS #1
```

APPEND è sempre una alternativa sicura ad **OUTPUT**, in quanto crea un nuovo file se non ne esiste già uno con il nome specificato. Ad esempio, qualora un file chiamato PREZZI.DAT non si trovi sul disco, l'istruzione dell'esempio precedente creerebbe un nuovo file con quel nome.

3.4.5.5 Alternative per l'immissione di dati in un file ad accesso sequenziale

Tutti gli esempi precedenti utilizzano l'istruzione **WRITE #** per scrivere i record in un file ad accesso sequenziale. Esiste, tuttavia, un'istruzione alternativa a **WRITE #**: **PRINT #**.

Il modo migliore per spiegare la differenza tra queste due istruzioni per la memorizzazione dei dati è quello di esaminare il contenuto di un file creato con entrambe. Il breve frammento seguente apre un file chiamato TEST.DAT, poi vi inserisce due volte lo stesso record, una volta con **WRITE #** e una volta con **PRINT #**. Dopo aver eseguito questo programma, è possibile esaminare il contenuto del file TEST.DAT con i comandi DOS TYPE:

```
OPEN "TEST.DAT" FOR OUTPUT AS #1
Nome$ = "Benso, Camillo"
Minist$ = "Esteri"
Livello = 10
Eta = 50
WRITE #1, Nome$, Minist$, Livello, Eta
PRINT #1, Nome$, Minist$, Livello, Eta
CLOSE #1
```

Output

```
"Benso, Camillo","Esteri",10,50
Benso, Camillo               Esteri         10             50
```

Nel record memorizzato con **WRITE #**, delle virgole delimitano esplicitamente ciascun campo del record, ed è racchiusa tra virgolette ogni espressione a stringa. **PRINT #**, invece, ha scritto sul file una copia del record esattamente nel modo in cui questo verrebbe visualizzato sullo schermo da una semplice istruzione **PRINT**. Le virgole nell'istruzione **PRINT #** significano "avanzare alla zona di visualizzazione successiva" (esiste una zona di visualizzazione ogni 14 spazi, a partire dall'inizio di una riga), e le espressioni a stringa non vengono racchiuse tra virgolette.

A questo punto, ci si potrebbe chiedere quale sia la differenza tra queste due istruzioni di output, se si esclude l'aspetto dei dati all'interno del file. La differenza si rivela quando il programma legge i dati dal file tramite un'istruzione **INPUT #**. Nell'esempio seguente, il programma legge il record memorizzato con **WRITE #** e ne visualizza i valori dei campi senza alcun problema:

```
OPEN "TEST.DAT" FOR INPUT AS #1

' Legge il primo record e ne visualizza i valori
' dei campi:
INPUT #1, Nome$, Minist$, Livello, Eta
PRINT Nome$, Minist$, Livello, Eta
```

```
' Legge il secondo record e ne visualizza i valori
' dei campi:
INPUT #1, Nome$, Minist$, Livello, Eta
PRINT Nome$, Minist$, Livello, Eta

CLOSE #1
```

Output

```
Benso, Camillo                Esteri          10              50
```

Tuttavia, quando il programma tenta di leggere il successivo record memorizzato con **PRINT #**, viene visualizzato il messaggio di errore `Lettura oltre la fine del file`. Se il primo campo non è racchiuso tra doppie virgolette, l'istruzione **INPUT #** considera la virgola tra `Benso` e `Camillo` come delimitatore di campo, ed assegna, quindi, soltanto il cognome `Benso` alla variabile `Nome$`. **INPUT$** poi legge il resto della riga nella variabile `Minist$`. Dal momento che ora sono stati letti tutti i record, non rimane nulla da inserire nelle variabili `Livello` e `Eta`. Il risultato è il messaggio di errore `Lettura oltre la fine del file`.

Se si stanno memorizzando dei record che contengono espressioni a stringa e successivamente si vorranno leggere questi record con l'istruzione **INPUT #**, si segua una di queste due indicazioni:

- Utilizzare l'istruzione **WRITE #** per memorizzare i record.
- Se invece si utilizza l'istruzione **PRINT #**, ci si deve ricordare che tale istruzione non inserisce le virgole nei record per delimitare i campi, né racchiude le stringhe tra virgolette. E' necessario che l'utente stesso inserisca questi separatori di campo nell'istruzione **PRINT #**.

Ad esempio, è possibile ovviare ai problemi esposti qui sopra utilizzando **PRINT #** con delle virgolette che delimitano esplicitamente ogni campo contenente un'espressione a stringa, come indicato nell'esempio seguente:

Esempio

```
' 34 è il codice ASCII delle doppie virgolette:

DV$ = CHR$(34)

' Con tutte e quattro le istruzioni successive, i campi a
' stringa dei record vengono trascritti al file tra doppie
' virgolette:
```

```
PRINT #1, DV$ Nome$ DV$ DV$ Minist$ DV$ Livello Eta
PRINT #1, DV$ Nome$ DV$;DV$ Minist$ DV$;Livello;Eta
PRINT #1, DV$ Nome$ DV$,DV$ Minist$ DV$,Livello,Eta
WRITE #1, Nome$, Minist$, Livello, Eta
```

Output al file

```
"Benso, Camillo""Esteri" 10  50
"Benso, Camillo""Esteri" 10  50
"Benso, Camillo"            "Esteri"       10             50
"Benso, Camillo","Esteri",10,50
```

3.4.5.6 Alternative per la lettura di dati da un file ad accesso sequenziale

Nelle sezioni precedenti, **INPUT #** viene utilizzato per leggere un record (una riga di dati da un file) assegnando i diversi campi del record alle variabili elencate dopo **INPUT #**. Questa sezione esamina alternative per la lettura di dati da un file ad accesso sequenziale, sia come record (**LINE INPUT #**) che come sequenze non formattate di byte (**INPUT$**).

L'istruzione LINE INPUT

Con l'istruzione **LINE INPUT #**, il programma può leggere una riga di testo esattamente come appare in un file, senza considerare eventuali virgole o virgolette come delimitatori di campo. Ciò si rivela particolarmente utile nei programmi che operano con file di testo ASCII.

L'istruzione **LINE INPUT #** legge una riga intera (fino ad una sequenza di a capo-interlinea) da un file ad accesso sequenziale, assegnandone il valore ad una variabile a stringa.

Esempi

Il breve programma seguente legge ciacuna riga dal file CAP1.TXT e la rivisualizza sullo schermo:

```
' Apre CAP1.TXT per la lettura sequenziale:
OPEN "CAP1.TXT" FOR INPUT AS #1

' Continua a leggere una riga per volta dal file finché
' non ne raggiunge la fine:
DO UNTIL EOF(1)
```

```
    ' Legge una intera riga e ne assegna il valore
    ' alla variabile BufferRiga$:
    LINE INPUT #1, BufferRiga$

    ' Visualizza la riga sullo schermo:
    PRINT BufferRiga$

LOOP
```

Il programma precedente viene facilmente modificato in programma di utilità per la copia di file, il quale, piuttosto che visualizzarla sullo schermo, stampa ogni riga letta dal file di input su un altro file:

```
' Ottiene i nomi dei file di input e output:

INPUT "File da copiare: ", File1$
IF File1$ = "" THEN END
INPUT "File su cui copiare: ", File2$
IF File2$ = "" THEN END

' Apre il primo file per la lettura sequenziale:
OPEN File1$ FOR INPUT AS #1

' Apre il secondo file per la scrittura sequenziale:
OPEN File2$ FOR OUTPUT AS #2

' Continua a leggere una riga per volta dal primo
' file finché non ne raggiunge la fine:
DO UNTIL EOF(1)

    ' Legge una intera riga dal primo file ne assegna
    ' il valore alla variabile BufferRiga$:
    LINE INPUT #1, BufferRiga$

    ' Scrive la riga nel secondo file:
    PRINT #2, BufferRiga$

LOOP
```

La funzione INPUT$

Un altro modo per leggere i dati dai file ad accesso sequenziale (e, in realtà, da qualsiasi file) consiste nell'utilizzare la funzione **INPUT$**. Mentre **INPUT #** e **LINE INPUT #** leggono da un file ad accesso sequenziale una riga alla volta, **INPUT$** legge un determinato numero di caratteri da un file, come mostrano gli esempi successivi:

Istruzione	*Azione*
`X$ = INPUT$(100, #1)`	Legge 100 caratteri dal file numero 1 e li assegna tutti alla variabile a stringa `X$`.
`Test$ = INPUT$(1, #2)`	Legge un solo carattere dal file numero 2 e lo assegna alla variabile a stringa `Test$`.

La funzione **INPUT$** senza il numero del file legge sempre l'input dalla periferica standard (di solito la tastiera).

La funzione **INPUT$** esegue l'input "binario"; vale a dire, legge un file come una sequela non formattata di caratteri. Ad esempio, non considera una sequenza di a capo-interlinea come segnalazione della fine di una operazione di input. Pertanto, **INPUT$** rappresenta l'alternativa migliore per far leggere al programma ogni carattere in un file, o per far leggere un file binario o un file non ASCII.

Esempio

Il programma seguente copia sullo schermo il dato file binario, visualizzandone solamente caratteri alfanumerici e di punteggiatura nell'intervallo ASCII da 32 a 126, così come le tabulazioni, gli a capo e le interlinee:

```
' 9 è il valore ASCII della tabulazione, 10 della
' interlinea, e 13 dell'a capo
CONST NRIGA = 10, ACAPO = 13, CARTAB = 9

INPUT "File da elaborare: ", File1$
IF File1$ = "" THEN END

OPEN File1$ FOR INPUT AS #1

DO UNTIL EOF(1)
   Car$ = INPUT$(1, #1)
   ValCar = ASC(Car$)
   SELECT CASE ValCar
      CASE 32 TO 126
         PRINT Car$;
```

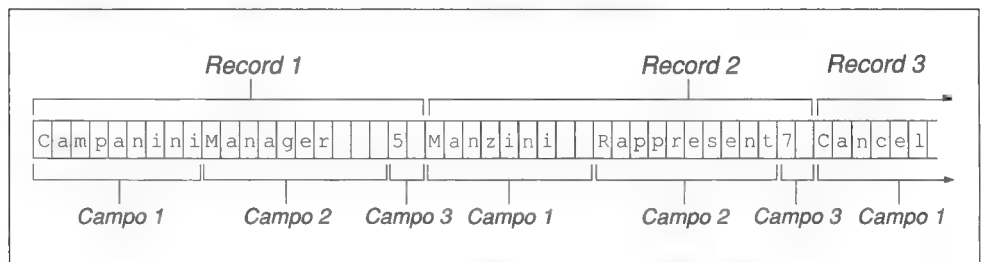
```
        CASE CARTAB, ACAPO
           PRINT Car$;
        CASE NRIGA
           IF ValCarPrec <> ACAPO THEN PRINT Car$;
        CASE ELSE
           ' Il carattere non è tra quelli da trascrivere.
     END SELECT
     ValCarPrec = ValCar
  LOOP
```

3.4.6 Utilizzo di file ad accesso casuale

Questa sezione spiega come i record vengono organizzati in file di dati ad accesso casuale e, quindi, come leggere i dati e come scriverli in un file aperto per l'accesso casuale.

3.4.6.1 Record in file ad accesso casuale

I record ad accesso casuale vengono memorizzati in maniera differente dai record ad accesso sequenziale. Ciascun record ad accesso casuale viene definito con una lunghezza fissa, come pure ciascun campo all'interno di esso. Queste lunghezze fisse determinano il punto in cui comincia e termina un record o un campo, poiché non ci sono virgole per separare i campi, nè sequenze di a capo-interlinea tra i record. L'illustrazione 3.3 mostra come possono essere registrati tre record in un file ad accesso casuale.

Illustrazione 3.3 Record in un file ad accesso casuale

Per i record che contengono numeri, l'utilizzo di file ad accesso casuale piuttosto che file ad accesso sequenziale risparmia dello spazio sul disco, in quanto i file ad accesso sequenziale memorizzano i numeri come una sequenza di caratteri ASCII che rappresentano ciascuna cifra, mentre i file ad accesso casuale memorizzano i numeri in un formato binario compresso.

Ad esempio, il numero 17000 viene rappresentato in un file ad accesso sequenziale con cinque byte, uno per ciascuna cifra. Se il numero 17000 è memorizzato in un campo a intero di un record ad accesso casuale, esso occupa solo due byte dello spazio del disco.

Generalmente, i numeri interi nei file ad accesso casuale occupano due byte; gli interi lunghi e i numeri in precisione semplice occupano quattro byte, mentre i numeri in precisione doppia ne occupano otto.

3.4.6.2 Aggiunta di dati in un file ad accesso casuale

Per scrivere un programma che aggiunge dati in un file ad accesso casuale, si devono seguire le seguenti indicazioni:

1. Definire i campi di ciascun record.
2. Aprire il file in modalità ad accesso casuale specificando la lunghezza dei record.
3. Leggere i dati del nuovo record e memorizzarlo nel file.

Queste indicazioni sono decisamente più semplici ora rispetto a quelle del BASICA, come mostrano gli esempi successivi.

Definizione dei campi

Un record può essere definito con l'istruzione **TYPE...END TYPE**, la quale permette di creare un tipo di dati misto, che combina elementi numerici con elementi a stringa. Questo metodo è più vantaggioso rispetto al precedente, che impostava i record per mezzo di un'istruzione **FIELD**, la quale richiedeva che ciascun campo fosse definito come una stringa. Definendo un record con **TYPE...END TYPE**, si può evitare l'utilizzo delle funzioni che convertono i dati numerici in stringhe (**MK***tipo***$**, **MKSMBF$** e **MKDMBF$**) e le stringhe in dati numerici (**CV***tipo*, **CVSMBF** e **CVDMBF**).

I frammenti che seguono mettono a confronto questi due metodi di definizione dei record:

- Record definito con **TYPE...END TYPE**:

```
' Definisce il tipo TipoRecord:
TYPE TipoRecord
   Nome AS STRING * 30
   Eta AS INTEGER
   Stipendio AS SINGLE
END TYPE
```

```
' Dichiara la variabile VarRecord
' del tipo dati TipoRecord:
DIM VarRecord AS TipoRecord
.
.
.
```

- Record definito con **FIELD**:

```
' Definisce le lunghezze dei campi nel buffer
' di transito:
FIELD #1, 30 AS Nome$, 2 AS Eta$, 4 AS Stipendio$
.
.
.
```

Apertura del file e specificazione della lunghezza dei record

Dato che la lunghezza dei record ad accesso casuale è costante, è necessario specificarla, altrimenti viene fissata al valore predefinito di 128 byte.

Per specificare la lunghezza dei record, utilizzare la clausola **LEN =** in un'istruzione **OPEN**. I due successivi frammenti mostrano come utilizzare **LEN =**.

- Specifica della lunghezza di un record definito con l'istruzione **TYPE...END TYPE**:

```
.
.
.
' Apre il file ad accesso casuale e specifica che la
' lunghezza dei record è uguale alla lunghezza della
' variabile VarRecord:
OPEN "DIPEND.DAT" FOR RANDOM AS #1 LEN = LEN(VarRecord)
.
.
.
```

- Specifica della lunghezza di un record definito con **FIELD**:

```
.
.
.
' Apre il file ad accesso casuale e specifica la
' lunghezza dei record:
OPEN "DIPEND.DAT" FOR RANDOM AS #1 LEN = 36
.
.
.
```

Come si può vedere, utilizzando **FIELD**, è necessario sommare la lunghezza di ciascun campo (`Nome$` è di 30 byte, `Eta$` è di 2 byte, `Stipendio$` è di 4 byte; per cui il record è lungo 30 + 2 + 4 o 36 byte). Con **TYPE...END TYPE**, invece di fare questi calcoli, si può semplicemente utilizzare la funzione **LEN** per calcolare la lunghezza della variabile creata per contenere i record (in questo caso, `VarRecord`).

Immissione di dati e memorizzazione del record

E' possibile immettere i dati direttamente negli elementi di un record definito dall'utente, senza preoccuparsi di allineare l'input a destra o a sinistra del campo con **LSET** o **RSET**. Si confrontino i due frammenti successivi, che sono la continuazione degli esempi precedenti, per vedere quant'è più breve questo secondo metodo.

- Immissione e memorizzazione dei dati di un record ad accesso casuale con **TYPE...END TYPE**:

```
.
.
.
' Legge i dati:
INPUT "Nome e cognome: ", VarRecord.Nome
INPUT "Età: ", VarRecord.Eta
INPUT "Stipendio: ", VarRecord.Stipendio

' Memorizza il record:
PUT #1, ,VarRecord
.
.
.
```

- Immissione e memorizzazione dei dati di un record ad accesso casuale con **FIELD**:

```
.
.
.
' Legge i dati:
INPUT "Nome e cognome: ", Nm$
INPUT "Età: ", ValEt%
INPUT "Stipendio: ", ValStip!
```

```
' Allinea i dati a sinistra nei campi del buffer di
' transito, usando le funzioni MKI$ e MKS$ per convertire
' i numeri in stringhe per il file:
LSET Nome$ = Nm$
LSET Eta$ = MKI$(ValEt%)
LSET Stipendio$ = MKS$(ValStip!)

' Memorizza il record:
PUT #1
.
.
.
```

Sintesi

Il programma che segue utilizza tutti i metodi delineati fin qui – per la definizione dei campi, la specifica della lunghezza dei record, e l'inserzione e memorizzazione dei dati di input – per aprire un file di dati ad accesso casuale, MAGAZ.DAT, e aggiungervi dei record.

```
DEFINT A-Z

' Definisce la struttura per i record del file ad
' accesso casuale:
TYPE UnitaInMag
   NumPezzo AS STRING * 6
   Descrizione AS STRING * 20
   PrezzoUnitario AS SINGLE
   Quantita AS INTEGER
END TYPE

' Definisce una variabile (RecordMag) usando il tipo
' UnitaInMag:
DIM RecordMag AS UnitaInMag

' Apre il file ad accesso casuale, specificando come
' lunghezza dei record la lunghezza di RecordMag:
OPEN "MAGAZ.DAT" FOR RANDOM AS #1 LEN = LEN(RecordMag)
```

```
' Calcola il numero dei record nel file usando LOF()
' perché i nuovi record vengano aggiunti alla fine:
NumDeiRecord = LOF(1) \ LEN(RecordMag)

' Aggiunge record al file dai dati digitati:
DO

   ' Legge i dati dalla tastiera e li immette nei
   ' vari campi della variabile RecordMag:
   INPUT "Numero del pezzo: ", RecordMag.NumPezzo
   INPUT "Descrizione     : ", RecordMag.Descrizione
   INPUT "Prezzo unitario : ", RecordMag.PrezzoUnitario
   INPUT "Quantità        : ", RecordMag.Quantita

   NumDeiRecord = NumDeiRecord + 1

   ' Scrive i dati da RecordMag a un nuovo record nel file:
   PUT #1, NumDeiRecord, RecordMag

   ' Controlla se vi sono altri record da aggiungere:
   INPUT "Aggiungere un altro record? (Sì/No) ", Risp$
LOOP UNTIL UCASE$(Risp$) = "N"

' Finito tutto; chiude il file ed esce:
CLOSE #1
END
```

Se il file MAGAZ.DAT esiste già, questo programma vi aggiunge ulteriori record senza sovrascrivere quelli già inseriti. La seguente istruzione è essenziale:

```
NumDeiRecord = LOF(1) \ LEN(RecordMag)
```

Il processo è il seguente:

1. La funzione `LOF(1)` calcola il numero totale di byte nel file MAGAZ.DAT. Se MAGAZ.DAT è nuovo o non contiene record, `LOF(1)` restituisce il valore 0.
2. La funzione `LEN(RecordMag)` calcola il numero di byte in un record. (`RecordMag` viene definito come avente la stessa struttura del tipo definito dall'utente `UnitaInMag`.)
3. Pertanto, il numero di record è uguale al totale dei byte del file diviso per i byte di un record.

Un record a lunghezza fissa presenta un'ulteriore vantaggio; poiché i record sono della stessa misura, è sempre possibile utilizzare la suddetta formula per calcolare il numero di record nel file. Ovviamente, ciò non funzionerebbe con un file ad accesso sequenziale, dal momento che i record ad accesso sequenziale possono avere lunghezze diverse.

3.4.6.3 Lettura di dati in sequenza

Con la tecnica delineata nella sezione precedente per calcolare il numero di record in un file ad accesso casuale, si può scrivere un programma che legga tutti i record contenuti in quel file.

Esempio

Il programma successivo legge i record in modo sequenziale (dal primo all'ultimo record memorizzato) nel file MAGAZ.DAT creato nella sezione precedente:

```
' Definisce una struttura di record per file ad
' accesso casuale:
TYPE UnitaInMag
   NumPezzo AS STRING * 6
   Descrizione AS STRING * 20
   PrezzoUnitario AS SINGLE
   Quantita AS INTEGER
END TYPE

' Dichiara una variabile (RecordMag) usando il tipo
' UnitaInMag:
DIM RecordMag AS UnitaInMag

' Apre il file ad accesso casuale:
OPEN "MAGAZ.DAT" FOR RANDOM AS #1 LEN = LEN(RecordMag)

' Calcola il numero dei record nel file:
NumDeiRecord = LOF(1) \ LEN(RecordMag)

' Legge i record dal file e invia i dati allo schermo:
FOR NumRecord = 1 TO NumDeiRecord

   ' Legge i dati dal successivo record nel file:
   GET #1, NumRecord, RecordMag
```

```
    ' Visualizza i dati sullo schermo:
    PRINT "Numero del pezzo: ", RecordMag.NumPezzo
    PRINT "Descrizione      : ", RecordMag.Descrizione
    PRINT "Prezzo unitario : ", RecordMag.PrezzoUnitario
    PRINT "Quantità         : ", RecordMag.Quantita

NEXT

' Finito tutto; chiude il file ed esce:
CLOSE #1
END
```

Non è necessario chiudere il file MAGAZ.DAT prima di leggerlo. L'apertura di un file ad accesso casuale con una singola istruzione **OPEN** consente di scrivere o leggere nel file.

3.4.6.4 Utilizzo dei numeri di record per l'estrazione dei record

E' possibile leggere qualsiasi record da un file ad accesso casuale specificandone il numero in un'istruzione **GET**. Si può scrivere in un qualsiasi record in un file ad accesso casuale, specificandone il numero in un'istruzione **PUT**. Questo è uno dei maggiori vantaggi che hanno i file ad accesso casuale rispetto a quelli ad accesso sequenziale, dal momento che questi ultimi non consentono un accesso diretto ad un record specifico.

Il programma esempio INDICI.BAS, riportato nella sezione 3.6.2, dimostra una tecnica che permette di trovare con rapidità un record particolare: la ricerca in un indice dei numeri di record.

Esempio

Il seguente brano illustra l'uso di **GET** con un numero di record:

```
DEFINT A-Z          ' Il tipo predefinito delle varabili
                    ' è intero.
CONST FALSO = 0, VERO = NOT FALSO

TYPE UnitaInMag
   NumPezzo AS STRING * 6
   Descrizione AS STRING * 20
   PrezzoUnitario AS SINGLE
   Quantita AS INTEGER
END TYPE
```

```
DIM RecordMag AS UnitaInMag

OPEN "MAGAZ.DAT" FOR RANDOM AS #1 LEN = LEN(RecordMag)

NumDeiRecord = LOF(1) \ LEN(RecordMag)
UnAltroRecord = VERO

DO
   PRINT "Digitare il numero del record da ";
   PRINT " visualizzare (oppure 0 per uscire): ";
   INPUT "", NumRecord

   IF NumRecord > 0 AND NumRecord < NumDeiRecord THEN

      ' Legge il record corrispondente al numero
      ' digitato e lo assegna a RecordMag:
      GET #1, NumRecord, RecordMag

      ' Visualizza il record:
      .
      .
      .

   ELSEIF NumRecord = 0 THEN
      UnAltroRecord = FALSO
   ELSE
      PRINT "Il valore digitato è fuori dall'intervallo."
   END IF
LOOP WHILE UnAltroRecord
END
```

3.4.7 I/O su file binari

L'accesso binario costituisce – dopo l'accesso casuale e sequenziale – il terzo modo per leggere o scrivere dati in un file.

Per aprire un file per I/O binario, si utilizza l'istruzione:

OPEN *file* **FOR BINARY AS #***numfile*

Con l'accesso binario si leggono direttamente i byte di qualsiasi file, non solo di un file ASCII. Pertanto si possono leggere o modificare i file non memorizzati in formato ASCII, come i file di Microsoft Word o i file eseguibili (.EXE).

I file aperti per l'accesso binario sono considerati come una sequenza non formattata di byte. Sebbene si possa leggere o scrivere un record (una variabile dichiarata come tipo definito dall'utente) su un file aperto in modalità binaria, lo stesso file non deve necessariamente essere organizzato all'interno in record a lunghezza fissa. Infatti, l'I/O di tipo binario non si occupa affatto dei record, a meno che non si considera ciascun byte di un file come un record a parte.

3.4.7.1 Confronto tra accesso binario ed accesso casuale

Le modalità di accesso binario ed accesso casuale hanno in comune la caratteristica di poter sia leggere che scrivere su un file dopo una sola istruzione **OPEN**. (L'accesso binario quindi differisce dall'accesso sequenziale, dove è necessario chiudere un file e riaprirlo se si vuole passare dalla lettura alla scrittura.) Inoltre, analogamente all'accesso casuale, l'accesso binario permette di spostarsi avanti e indietro all'interno di un file aperto. L'accesso binario supporta anche le stesse istruzioni utilizzate per leggere e scrivere nei file ad accesso casuale:

{GET | PUT} [#]*numerofile*, **[***posizione***]**, *variabile*

Qui, *variabile* può essere di qualunque tipo, compresa la stringa a lunghezza variabile o un tipo definito dall'utente, e *posizione* indica il punto nel file in cui avrà luogo la successiva operazione **GET** o **PUT**. (La *posizione* è relativa all'inizio del file; cioè, il primo byte nel file ha *posizione* uno, il secondo ha *posizione* due, e così via.) Se si omette l'argomento *posizione*, le successive operazioni **GET** e **PUT** spostano il puntatore del file in avanti, dal primo byte all'ultimo.

L'istruzione **GET** legge dal file un numero di byte uguale alla lunghezza della *variabile*. Analogamente, l'istruzione **PUT** scrive nel file un numero di byte uguale alla lunghezza della *variabile*. Per esempio, se la *variabile* è di tipo intero, **GET** vi assegnerà due byte, se la *variabile* è del tipo in precisione semplice, **GET** vi assegnerà quattro byte. Pertanto, se in un'istruzione **GET** o **PUT** non viene specificato un argomento *posizione*, il puntatore del file verrà spostato in avanti una distanza pari alla lunghezza della *variabile*.

L'istruzione **GET** e la funzione **INPUT$** sono gli unici modi per leggere i dati da un file aperto in modalità binaria.

L'istruzione **PUT$** è l'unico modo, invece, per scrivere in un file aperto in questa modalità.

L'accesso binario, a differenza dell'accesso casuale, permette di spostarsi alla posizione di qualunque byte del file e leggervi, o scrivervi, qualunque numero di byte desiderato. Al contrario, l'accesso casuale permette soltanto di spostarsi all'inizio di un record e di leggere un numero fisso di byte alla volta (la lunghezza di un record).

3.4.7.2 Posizionamento del puntatore del file con SEEK

Per spostare il puntatore del file ad un certo punto senza eseguire alcun I/O, utilizzare l'istruzione **SEEK**. La sintassi è

SEEK *numerofile*, *posizione*

Dopo un'istruzione **SEEK**, la successiva operazione di lettura o scrittura sul file aperto con *numerofile* inizierà dal byte specificato in *posizione*. La funzione **SEEK** è l'inversa dell'istruzione **SEEK**, e ha questa sintassi:

SEEK (*numerofile*)

La funzione **SEEK** indica la posizione del byte in cui inizierà la successiva operazione di lettura o scrittura. (Utilizzando l'I/O binario per accedere ad un file, le funzioni **LOC** e **SEEK** danno risultati analoghi, ma **LOC** restituisce la posizione dell'ultimo byte letto o scritto, mentre **SEEK** restituisce la posizione del successivo byte da leggere o da scrivere.)

L'istruzione e la funzione **SEEK** funzionano anche su file aperti per l'accesso sequenziale o casuale. Con l'accesso sequenziale, sia l'istruzione che la funzione si comportano allo stesso modo che con l'acceso binario; cioè, l'istruzione **SEEK** sposta il puntatore del file sul byte con la determinata posizione, e la funzione **SEEK** restituisce informazioni relative al successivo byte da leggere o scrivere.

Tuttavia, se un file viene aperto per l'accesso casuale, l'istruzione **SEEK** può spostare il puntatore del file solo sull'inizio di un record, e non su un byte all'interno di questo. Inoltre, quando viene utilizzata con i file ad accesso casuale, la funzione **SEEK** restituisce il numero del record successivo, piuttosto che la posizione del byte successivo.

Esempio

Il programma seguente apre una libreria Quick di QuickBASIC, e quindi legge e stampa i nomi delle procedure BASIC e altri simboli esterni contenuti nella libreria. Questo programma si trova nel file QLBDUMP.BAS sui dischi della confezione QuickBASIC.

```
' Questo programma stampa i nomi delle procedure in una
' libreria Quick.
DECLARE SUB DumpSim (InizSimb AS INTEGER, PosTstQ AS LONG)
```

```
TYPE TestExe                        ' Parte della testata .EXE
                                    ' di DOS
    altri1    AS STRING * 8        ' Altri dati sulla testata
    NParTes   AS INTEGER           ' Numero di paragrafi nella
                                    ' testata
    altri2    AS STRING * 10       ' Altri dati sulla testata
    IP        AS INTEGER           ' Valore IP iniziale
    CS        AS INTEGER           ' Valore CS iniziale (relativo)
END TYPE

TYPE TesQB                          ' Testata QLB
    StitQB    AS STRING * 6        ' Sottotitolo particolare a QB
    Sesamo    AS INTEGER           ' Parola che identifica i file
                                    ' libreria Quick
    InizSimb  AS INTEGER           ' Offset dalla testata al primo
                                    ' simbolo codice
    InizDati  AS INTEGER           ' Offset dalla testata al primo
                                    ' simbolo dati
END TYPE

TYPE SimQb                          ' Voce simbolica nella QuickLib
    Flag      AS INTEGER           ' Segnalatore di simboli
    InizioNom AS INTEGER           ' Offset nella tabella dei nomi
    altri     AS STRING * 4        ' Altri dati sulla testata
END TYPE

DIM TstE AS TestExe, TstQ AS TesQB, PosTstQ AS LONG

INPUT "Nome del file libreria Quick: ", NomeFile$
NomeFile$ = UCASE$(NomeFile$)
IF INSTR(NomeFile$, ".QLB") = 0 THEN _
   NomeFile$ = NomeFile$ + ".QLB"

INPUT "Nome del file di destinazione (o INVIO per lo ";
PRINT "schermo): ", FileDest$

FileDest$ = UCASE$(FileDest$)
IF FileDest$ = "" THEN FileDest$ = "CON"

OPEN NomeFile$ FOR BINARY AS #1
OPEN FileDest$ FOR OUTPUT AS #2
```

```
GET #1, , TstE                    ' Legge la testata del
                                  ' formato EXE.
PosTstQ = (TstE.NParTes + TstE.CS) * 16 + TstE.IP + 1

GET #1, PosTstQ, TstQ             ' Legge la testata del
                                  ' formato QuickLib.
IF TstQ.Sesamo <> &H6C75 THEN PRINT "Non è una _
   libreria Quick di QB": END

PRINT #2, "Simboli di codice:": PRINT #2,
DumpSim TstQ.InizSimb, PosTstQ          ' Esegue il dump dei
                                        ' simboli di codice.
PRINT #2,

PRINT #2, "Simboli di dati:": PRINT #2, ""
DumpSim TstQ.InizDati, PosTstQ          ' Esegue il dump dei
                                        ' simboli di dati.
PRINT #2,

END

SUB DumpSim (InizSimb AS INTEGER, PosTstQ AS LONG)
   DIM SimQlb AS SimQb
   DIM SimSucc AS LONG, SimCorr AS LONG

   ' Calcola la posizione della prima voce simbolica e
   ' la legge:
   SimSucc = PosTstQ + InizSimb
   GET #1, SimSucc, SimQlb

   DO
      SimSucc = SEEK(1)              ' Salva la posizione del
                                     ' simbolo successivo.
      SimCorr = PosTstQ + SimQlb.InizioNom
      SEEK #1, SimCorr               ' Usa SEEK per spostarsi al
                                     ' nome della voce simbolica
                                     ' corrente.
      Prospett$ = INPUT$(40, 1)      ' Legge una stringa della
                                     ' massima lunghezza valida,
                                     ' più un byte per il
                                     ' carattere nullo finale
                                     ' (CHR$(0)).
```

```
      ' Estrae il nome terminante nel carattere nullo:
      NomeS$ = LEFT$(Prospett$, INSTR(Prospett$, CHR$(0)))

      ' Stampa solo i nomi che non iniziano per "__", "$", o
      ' "b$", perché questi sono generalmente riservati:
      IF LEFT$(NomeS$, 2) <> "__" AND LEFT$(NomeS$, 1) <> _
         "$" AND UCASE$(LEFT$(NomeS$, 2)) <> "B$" THEN
         PRINT #2, "  " + NomeS$
      END IF

      GET #1, SimSucc, SimQlb     ' Legge una voce simbolica.
   LOOP WHILE SimQlb.Flag         ' Flag = 0 (FALSO) segnala
                                  ' la fine della tabella.
END SUB
```

3.5 Utilizzo di periferiche

Il Microsoft BASIC supporta l'I/O su periferica. Alcune periferiche del computer possono essere aperte per l'I/O allo stesso modo dei file di dati su disco. L'input o l'output su queste periferiche può essere ottenuto con le istruzioni e le funzioni elencate nella tabella 9.4, "Riassunto della istruzioni di I/O su file", con le eccezioni rilevate nella sezione 3.5.1, "Differenze tra I/O su periferiche e I/O su file". La tabella 3.1 elenca le periferiche supportate dal BASIC.

Tabella 3.1 Periferiche Supportate dal BASIC per I/O

Nome	*Periferica*	*Modalità di I/O supportata*
COM1:	Prima porta seriale	Input e Output
COM2:	Seconda porta seriale	Input e Output
CONS:	Schermo	Solo Output
KYBD:	Tastiera	Solo Input
LPT1:	Prima stampante	Solo Output
LPT2:	Seconda stampante	Solo Output
LPT3:	Terza stampante	Solo Output
SCRN:	Schermo	Solo Output

3.5.1 Differenze tra I/O su periferica e I/O su file

Alcune funzioni ed istruzioni utilizzate per l'I/O su file non sono consentite per I/O su periferica, e altre ancora hanno una funzione diversa. Ecco alcune differenze:

- Le periferiche CONS:, SCRN:, e LPT*n*: non possono essere aperte nelle modalità **INPUT** o **APPEND**.
- La periferica KYBD: non può essere aperta in modalità **OUTPUT** o **APPEND**.
- Le funzioni **EOF**, **LOC**, e **LOF** non possono essere utilizzate con le periferiche CONS:, KYBD:, LPT*n*: o SCRN:.
- Le funzioni **EOF**, **LOC**, e **LOF** si possono utilizzare con la periferica seriale **COM*n*:**; tuttavia, i valori che queste funzioni restituiranno avranno un significato diverso rispetto ai valori restituiti quando vengono utilizzate con file di dati. (Per chiarimenti relativi all'uso di queste funzioni con **COM*n*:**, si consulti la sezione 3.5.2.)

Esempio

Il programma successivo mostra come le periferiche LPT1: e SCRN: possono essere aperte per l'output, utilizzando la stessa sintassi dei file di dati. Questo programma legge tutte le righe del file selezionato dall'utente, quindi ne stampa le righe sullo schermo o sulla stampante a seconda della scelta dell'utente.

```
CLS
' Legge il nome del file da visualizzare:
INPUT "Nome del file da visualizzare: ", NomeFile$

' Se non viene digitato un nome, termina il programma;
' altrimenti, apre il file per la lettura (INPUT):
IF NomeFile$ = "" THEN END ELSE OPEN NomeFile$ FOR INPUT _
   AS #1

' Legge la scelta per l'output del file (stampante o
' schermo), e continua a sollecitare che l'utente prema
' "ST" o "SC":
DO
   ' Posiziona il cursore alla 2a riga, 1a colonna e
   ' visualizza il sollecito:
   LOCATE 2, 1, 1
   PRINT "Inviare l'output allo schermo (SC) o alla ";
   PRINT "stampante (ST) : ";
```

```
   ' Cancella i tre spazi dopo il sollecito:
   PRINT SPACE$(2);

   ' Riposiziona il cursore dopo il sollecito:
   LOCATE 2, 60, 1
   Scelta$ = UCASE$(INPUT$(2))          ' Legge la scelta.
   PRINT Scelta$
LOOP WHILE Scelta$ <> "SC" AND Scelta$ <> "ST"

' A seconda della scelta, apre o la stampante o
' lo schermo per l'output:
SELECT CASE Scelta$
   CASE "SC"
      OPEN "SCRN:" FOR OUTPUT AS #2
   CASE "ST"
      OPEN "LPT1:" FOR OUTPUT AS #2
      PRINT "Stampa del file in corso sulla stampante LPT1:"
END SELECT

' Imposta la larghezza della periferica scelta a 80 colonne:
WIDTH #2, 80

' Finché ci sono ancora righe nel file, legge una riga dal
' file e la invia alla periferica scelta:
DO UNTIL EOF(1)
   LINE INPUT #1, BufferRiga$
   PRINT #2, BufferRiga$
LOOP

CLOSE          ' Termina l'input dal file e l'output alla
               ' periferica.
END
```

3.5.2 Comunicazioni attraverso la porta seriale

L'istruzione **OPEN "COM*n*:"** (dove *n* può essere 1 o, se ci sono due porte seriali, 2) permette di aprire la porta o le porte seriali del computer per la comunicazione seriale (bit per bit) con altri computer o con unità periferiche quali i modem o le stampanti seriali. Ecco alcuni dei parametri che si possono determinare:

- Velocità di trasmissione di dati, misurata in "baud" (bit al secondo)
- Se e come rilevare gli eventuali errori di trasmissione
- Quanti bit di stop (1, 1,5 o 2) si devono utilizzare per segnalare la fine di un byte trasmesso
- Quanti bit in ogni byte di dati trasmesso o ricevuto costituiscono dati effettivi

Quando si apre la porta seriale per comunicazioni, viene messo da parte un buffer di input per contenere i byte che vengono letti dall'altra periferica. Ciò avviene perché, ad alte velocità di baud, i caratteri arrivano più velocemente di quanto possano essere elaborati. La dimensione predefinita per questo buffer è 512 byte, e può essere modificata con l'opzione **LEN** = *numbyte* dell'istruzione **OPEN "COM*n*:"**. I valori restituiti dalle funzioni **EOF**, **LOC** e **LOF** quando sono utilizzate con una periferica di comunicazione, restituiscono informazioni relative alla dimensione di questo buffer, come è illustrato nell'elenco seguente:

Funzione	*Informazione restituita*
EOF	Se vi sono o no caratteri da leggere in attesa nel buffer di input
LOC	Il numero dei caratteri in attesa nel buffer di input
LOF	La quantità di spazio libero (in byte) nel buffer di output

Poiché ogni carattere è potenzialmente un dato significativo, sia con **INPUT #** che **LINE INPUT #** si incontrano gravi inconvenienti nell'ottenere l'input da un'altra periferica. Ciò avviene perché **INPUT #** termina la lettura dei dati per una variabile quando incontra una virgola o una nuova riga (e, talvolta, uno spazio o doppie virgolette), e **LINE INPUT #** termina la lettura dei dati quando incontra un nuova riga. Ciò rende **INPUT$** la migliore funzione da utilizzare per ottenere dell'input da una periferica di comunicazione, dal momento che legge tutti i caratteri.

La riga seguente utilizza la funzione **LOC** per verificare la quantità di caratteri in attesa nel buffer provenienti dalla periferica di comunicazione aperta come file #1; essa utilizza la funzione **INPUT$** per leggere quei caratteri, assegnandoli alla variabile a stringa `ModemInput$`:

```
ModemInput$ = INPUT$(LOC(1), #1)
```

3.6 Programmi esempio

I programmi esempio elencati in questa sezione includono un programma di gestione dello schermo che stampa un calendario per qualunque mese di qualunque anno dal 1899 al 2099, un programma di I/O su file che crea e ricerca un indice dei numeri del record da un file ad accesso casuale, e un programma di comunicazione che permette al PC di comportarsi come un terminale, quando esso viene collegato ad un modem.

3.6.1 Calendario perpetuo (CAL.BAS)

Dopo aver richiesto all'utente di inserire un mese da 1 a 12 e un anno compreso tra il 1899 e il 2099, il seguente programma stampa il calendario per quel determinato mese ed anno. La procedura `AnnoBisest` fa le opportune modifiche al calendario per i mesi di un anno bisestile.

Istruzioni e funzioni utilizzate

Il programma usa le seguenti funzioni ed istruzioni di gestione dello schermo trattate nelle sezioni 3.1–3.3:

- **INPUT**
- **INPUT$**
- **LOCATE**
- **POS(0)**
- **PRINT**
- **PRINT USING**
- **TAB**

Listato del programma

Il programma del calendario perpetuo CAL.BAS è elencato qui sotto:

```
DEFINT A-Z                       ' Il tipo predefinito per le
                                 ' variabili è intero.

' Definisce un tipo di dati per i nomi dei mesi e il
' numero di giorni in ogni mese:
TYPE TipoMese
   NumerG AS INTEGER             ' Numero di giorni nel mese
   NomeM AS STRING * 9           ' Nome del mese
END TYPE
```

```
' Dichiara le procedure usate:
DECLARE FUNCTION AnnoBisest% (N%)
DECLARE FUNCTION OttInput% (Prompt$, Riga%, ValBasso%, _
   ValAlto%)

DECLARE SUB VisualizCal (Anno%, Mese%)
DECLARE SUB CalcolaMese (Anno%, Mese%, GiornoIniz%, _
   NumGiorni%)

DIM DatiMese (1 TO 12) AS TipoMese

' Inizializza le definizioni dei mesi dalle istruzioni DATA
' in fondo:
FOR I = 1 TO 12
   READ DatiMese(I).NomeM, DatiMese(I).NumerG
NEXT

' Ciclo principale; ripete se si desidera vedere altri mesi:
DO
   CLS

   ' Ottiene l'anno e il mese come input:
   Anno = OttInput("Anno (dal 1899 al 2099): ", 1, 1899, _
      2099)
   Mese = OttInput("Mese (da 1 a 12): ", 2, 1, 12)

   ' Visualizza il calendario:
   VisualizCal Anno, Mese

   ' Un'altra data?
   LOCATE 13, 1                         ' Posiziona su 13a riga,
                                        ' 1a colonna.
   PRINT "Un'altra data? (Sì/No)";      ' Tiene il cursore sulla
                                        ' stessa riga.
   LOCATE , , 1, 0, 13                  ' Attiva il cursore e lo
                                        ' fa alto un carattere.
   Risp$ = INPUT$(1)                    ' Attende la pressione
                                        ' di un tasto.
   PRINT Risp$                          ' Visualizza il tasto
                                        ' premuto.

LOOP WHILE UCASE$(Risp$) = "S"
END
```

```
' Dati per i mesi dell'anno:
DATA Gennaio, 31, Febbraio, 28, Marzo, 31, Aprile, 30,
DATA Maggio, 31, Giugno, 30, Luglio, 31, Agosto, 31
DATA Settembre, 30, Ottobre, 31, Novembre, 30, Dicembre, 31

' ====================== ANNOBISEST ======================
'  Determina se un anno è bisestile o no.
' ========================================================

FUNCTION AnnoBisest (N) STATIC

   ' Se l'anno è divisibile per 4 ma non per 100,
   ' oppure se è divisibile per 400, significa che
   ' è bisestile:
   AnnoBisest = (N MOD 4 = 0 AND N MOD 100 <> 0) OR (N MOD _
      400 = 0)
END FUNCTION

' ====================== CALCOLAMESE =====================
'  Calcola il primo giorno e il numero di giorni in un mese.
' ========================================================

SUB CalcolaMese (Anno, Mese, GiornoIniz, NumGiorni) STATIC
   SHARED DatiMese() AS TipoMese
   CONST BISEST = 366 MOD 7
   CONST NORMALE = 365 MOD 7

   ' Calcola il totale dei giorni (TotGiorni) dal 1/1/1899:

   ' Comincia dagli anni interi:
   TotGiorni = -1
   FOR I = 1899 TO Anno - 1
      IF AnnoBisest(I) THEN
         ' Se l'anno è bisestile, aggiungere 366 MOD 7:
         TotGiorni = TotGiorni + BISEST
      ELSE
         ' Se l'anno è normale, aggiungere 365 MOD 7:
         TotGiorni = TotGiorni + NORMALE
      END IF
   NEXT
```

```
   ' Poi, aggiungere i giorni dei mesi interi:
   FOR I = 1 TO Mese - 1
      TotGiorni = TotGiorni + DatiMese(I).NumerG
   NEXT

   ' Imposta il numero di giorni del mese desiderato:
   NumGiorni = DatiMese(Mese).NumerG

   ' Compensa se l'anno desiderato è bisestile:
   IF AnnoBisest(Anno) THEN

      ' Se è dopo febbraio, aggiunge uno al numero totale
      ' dei giorni:
      IF Mese > 2 THEN
         TotGiorni = TotGiorni + 1

      ' Se è febbraio, aggiunge uno al numero dei giorni
      ' del mese:
      ELSEIF Mese = 2 THEN
         NumGiorni = NumGiorni + 1

      END IF
   END IF

   ' 1/1/1899 era domenica (-1), quindi calcolando
   ' "TotGiorni MOD 7" si ottiene il giorno della settimana
   ' (lunedì = 0, martedì = 1, e così via) del primo giorno
   ' del mese desiderato:
   GiornoIniz = TotGiorni MOD 7
END SUB

' ======================= OTTINPUT =========================
'  Sollecita l'input, quindi verifica se è compreso
'  nell'intervallo.
' ==========================================================

FUNCTION OttInput (Prompt$, Riga, ValBasso, ValAlto) STATIC

   ' Posiziona il prompt alla riga specificata, attiva il
   ' cursore e lo fa alto un carattere:
   LOCATE Riga, 1, 1, 0, 13
   PRINT Prompt$;
```

```
    ' Salva la posizione della colonna:
    Colonna = POS(0)

    ' Legge un valore finché non è compreso nell'intervallo:
    DO
       LOCATE Riga, Colonna    ' Posiziona il cursore dopo
                               ' il prompt.
       PRINT SPACE$(10)        ' Cancella ciò che potrebbe
                               ' trovarvi.
       LOCATE Riga, Colonna    ' Riposiziona il cursore dopo
                               ' il prompt.
       INPUT "", Valore        ' Legge il valore (e scarta
                               ' il prompt).
    LOOP WHILE (Valore < ValBasso OR Valore > ValAlto)

    ' Restituisce un input valido come valore della funzione:
    OttInput = Valore

END FUNCTION

' ====================== VISUALIZCAL =======================
'  Visualizza il calendario del mese nell'anno specificato.
' ==========================================================

SUB VisualizCal (Anno, Mese) STATIC
SHARED DatiMese() AS TipoMese

    ' Calcola il primo giorno (L Ma Me...) e il numero di
    ' giorni nel mese:
    CalcolaMese Anno, Mese, GiornoIniz, NumGiorni
    CLS
    Testata$ = RTRIM$(DatiMese(Mese).NomeM) + "," + _
       STR$(Anno)

    ' Calcola la posizione sullo schermo per il mese e
    ' l'anno:
    MargineSin = (35 - LEN(Testata$)) \ 2

    ' Stampa la testata:
    PRINT TAB(MargineSin); Testata$
    PRINT
    PRINT " L    Ma   Me   G    V    S    D"
    PRINT
```

```
    ' Ricalcola e stampa la tabulazione al primo
    ' giorno del mese (L Ma Me...):
    MargineSin = 5 * GiornoIniz + 1
    PRINT TAB(MargineSin);

    ' Stampa i giorni del mese:
    FOR I = 1 TO NumGiorni
       PRINT USING "##   "; I;

       ' Avanza alla riga successiva quando il cursore
       ' passa la colonna 32:
       IF POS(0) > 32 THEN PRINT
    NEXT

END SUB
```

Output

```
          Gennaio, 2001

   L    Ma   Me   G    V    S    D

   1    2    3    4    5    6    7
   8    9   10   11   12   13   14
  15   16   17   18   19   20   21
  22   23   24   25   26   27   28
  29   30   31


  Un'altra data? (Sì/No)
```

3.6.2 Indicizzazione di un file ad accesso casuale (INDICI.BAS)

Il programma seguente utilizza una tecnica di indicizzazione per memorizzare ed estrarre i record di un file ad accesso casuale. Ciascun elemento della matrice `Indice` è composto da due parti: un campo stringa (`NumPezzo`) e un campo intero (`NumRecord`). Questa matrice viene posta in ordine alfabetico sul campo `NumPezzo`, il che permette di cercare rapidamente nella matrice un determinato numero di pezzo, utilizzando una ricerca di tipo binario.

La matrice `Indice` funziona in modo molto simile all'indice di un libro. Per cercare in un libro le pagine relative ad un argomento, basta cercarne la voce nell'indice, e accanto alla voce verrà indicata la pagina. Analogamente, questo programma cerca un numero di pezzo nella matrice degli indici `Indice`, ordinata in ordine alfabetico. Una volta trovato tale numero, il numero del record associato al campo `NumRecord` indica il record contenente tutte le informazioni relative a quel pezzo.

Istruzioni e funzioni utilizzate

Questo programma utilizza le seguenti funzioni ed istruzioni per accedere ai file ad accesso casuale:

- **TYPE...END TYPE**
- **OPEN...FOR RANDOM**
- **GET #**
- **PUT #**
- **LOF**

Listato del programma

Ecco il listato del programma INDICI.BAS, che indicizza un file ad accesso casuale:

```
DEFINT A-Z

' Definisce le costanti simboliche usate globalmente nel
' programma:
CONST FALSO = 0, VERO = NOT FALSO

' Definisce la struttura di record per i record del file
' ad accesso casuale:
```

```
TYPE UnitaInMag
   NumPezzo  AS STRING * 6
   Descrizione AS STRING * 20
   PrezzoUnitario   AS SINGLE
   Quantita    AS INTEGER
END TYPE

' Definisce una struttura di record per ciascun elemento
' dell'indice:
TYPE TipoIndice
   NumRecord AS INTEGER
   NumPezzo   AS STRING * 6
END TYPE

' Dichiara le procedure da chiamare:
DECLARE FUNCTION Filtro$(Prompt$)
DECLARE FUNCTION TrovaRecord%(NumPezzo$, VarRecord AS _
   UnitaInMag)
DECLARE SUB AggRecord(VarRecord AS UnitaInMag)
DECLARE SUB LeggiRecord(VarRecord AS UnitaInMag)
DECLARE SUB StampaRecord(VarRecord AS UnitaInMag)
DECLARE SUB OrdinaIndice()
DECLARE SUB VisNumPezzo()

' Definisce un buffer (usando il tipo UnitaInMag) e
' definisce e dimensiona la matrice dell'indice:
DIM RecordMag AS UnitaInMag, Indice(1 TO 100) AS TipoIndice

' Apre il file ad accesso casuale:
OPEN "MAGAZ.DAT" FOR RANDOM AS #1 LEN = LEN(RecordMag)

' Calcola il numero dei record nel file:
NumDeiRecord = LOF(1) \ LEN(RecordMag)

' Se ci sono record, li legge e crea l'indice:
IF NumDeiRecord <> 0 THEN
   FOR NumRecord = 1 TO NumDeiRecord

      ' Legge i dati dai nuovi record nel file:
      GET #1, NumRecord, RecordMag
```

```
      ' Immette il numero del pezzo e il numero del record
      ' nell'indice:
      Indice(NumRecord).NumRecord = NumRecord
      Indice(NumRecord).NumPezzo = RecordMag.NumPezzo
   NEXT

   OrdinaIndice            ' Ordina l'indice secondo il numero
                           ' del pezzo.
END IF

DO                         ' Ciclo del menu principale.
   CLS
   PRINT "(A)ggiungi dei record."
   PRINT "(V)isualizza dei record."
   PRINT "(E)sci dal programma."
   PRINT
   LOCATE , , 1
   PRINT "Digitare l'azione desiderata (A, V, o E): ";
   ' Cicla finché l'utente non digita A, V, o E:
   DO
      Scelta$ = UCASE$(INPUT$(1))
   LOOP WHILE INSTR("AVE", Scelta$) = 0

   ' Dirama secondo la scelta fatta:
   SELECT CASE Scelta$
      CASE "A"
         AggRecord RecordMag
      CASE "V"
         IF NumDeiRecord = 0 THEN
            PRINT : PRINT "Nessun record nel file. ";
            PRINT "Premere un tasto per continuare.";
            Pausa$ = INPUT$(1)
         ELSE
            LeggiRecord RecordMag
         END IF
      CASE "E"             ' Esce dal programma
   END SELECT
LOOP UNTIL Scelta$ = "E"

CLOSE #1                   ' Tutto fatto, chiude il file
                           ' ed esce.
END
```

```
' ====================== AggRecord =======================
'  Aggiunge record al file dai dati digitati.
' ==========================================================

SUB AggRecord (VarRecord AS UnitaInMag) STATIC
   SHARED Indice() AS TipoIndice, NumDeiRecord
   DO
      CLS
      INPUT "Numero del pezzo: ", VarRecord.NumPezzo
      INPUT "Descrizione     : ", VarRecord.Descrizione
      VarRecord.PrezzoUnitario = VAL(Filtro$ _
         ("Prezzo unitario   : "))
      VarRecord.Quantita = VAL(Filtro$_
         ("Quantità          : "))
      NumDeiRecord = NumDeiRecord + 1

      PUT #1, NumDeiRecord, VarRecord

      Indice(NumDeiRecord).NumRecord = NumDeiRecord
      Indice(NumDeiRecord).NumPezzo = VarRecord.NumPezzo
      PRINT : PRINT "Aggiungere un altro record? (Sì/No)";
      OK$ = UCASE$(INPUT$(1))
   LOOP WHILE OK$ = "S"
   OrdinaIndice                       ' Riordina l'indice.
END SUB

' ======================== Filtro =========================
'  Elimina tutti i caratteri non numerici da una stringa
'  e restituisce la stringa filtrata.
' ==========================================================

FUNCTION Filtro$(Prompt$) STATIC
   ValTemp2$ = ""
   PRINT Prompt$;                     ' Stampa il prompt
                                      ' passato.
   INPUT "", ValTemp1$                ' Legge il numero come
                                      ' stringa.
   LungStringa = LEN(ValTemp1$)       ' Ottiene la lunghezza
                                      ' della stringa.
   FOR I% = 1 TO LungStringa          ' Esamina la stringa un
      Car$ = MID$(ValTemp1$, I%, 1)   ' carattere per volta.
```

```
         ' Il carattere fa parte di un numero (cioè è una
         ' cifra o una virgola decimale)?  Se sì, la aggiunge
         ' alla fine della nuova stringa:
         IF INSTR(".0123456789", Car$) > 0 THEN
            ValTemp2$ = ValTemp2$ + Car$
         ' Se no, controlla se è una "l" minuscola, perché
         ' un utente abituato a battere a macchina potrebbe
         ' digitarla al posto della cifra "1":
         ELSEIF Car$ = "l" THEN
            ValTemp2$ = ValTemp2$ + "1" ' Cambia la "l" in "1".
         END IF
      NEXT I%
      Filtro$ = ValTemp2$                ' Restituisce la
                                         ' stringa filtrata.
   END FUNCTION

   ' ====================== LeggiRecord =======================
   '  LeggiRecord prima chiama VisNumPezzo, che stampa
   '  il menu dei numeri di pezzo sullo schermo in alto.
   '  LeggiRecord poi sollecita dall'utente l'immissione di un
   '  numero di pezzo. Infine, chiama le procedure TrovaRecord
   '  e StampaRecord per trovare e stampare il record scelto.
   ' ==========================================================

   SUB LeggiRecord(VarRecord AS UnitaInMag) STATIC
      CLS
      VisNumPezzo          ' Chiama la SUBRoutine VisNumPezzo.

      ' Stampa sullo schermo in basso i dati dei record scelti:
      DO
         PRINT "Digitare uno dei numeri di pezzo elencati in ";
         INPUT "alto (o U per uscire), e premere <INVIO>: _
            ", Pezzo$
         IF UCASE$(Pezzo$) <> "U" THEN
            IF TrovaRecord%(Pezzo$, VarRecord) THEN
               StampaRecord VarRecord
            ELSE
               PRINT "Pezzo non trovato."
            END IF
         END IF
         PRINT STRING$(40, "_")
      LOOP WHILE UCASE$(Pezzo$) <> "U"
```

```
   VIEW PRINT          ' Reimposta la finestra di testo
                       ' all'intero schermo.
END SUB

' ===================== OrdinaIndice ======================
'  Ordina l'indice secondo il numero del pezzo.
' =========================================================

SUB OrdinaIndice STATIC
   SHARED Indice() AS TipoIndice, NumDeiRecord

   ' Imposta l'offset per il confronto alla metà del
   ' numero dei record nell'indice:
   Offset = NumDeiRecord \ 2

   ' Cicla finché l'offset non diventa zero:
   DO WHILE Offset > 0
      Limite = NumDeiRecord - Offset
      DO

         ' Presume non vi siano stati scambi con questo
         ' offset:
         Scambi = FALSO

         ' Confronta elementi e scambia quelli non in
         ' ordine:
         FOR I = 1 TO Limite
            IF Indice(I).NumPezzo > Indice(I + Offset) _
               .NumPezzo THEN
               SWAP Indice(I), Indice(I + Offset)
               Scambi = I
            END IF
         NEXT I

         ' Al successivo passaggio, ordina solo fino al
         ' punto del precedente scambio:
         Limite = Scambi
      LOOP WHILE Scambi

      ' Nessuno scambio al precedente offset; dimezza
      ' l'offset:
      Offset = Offset \ 2
   LOOP
END SUB
```

```
' ====================== StampaRecord ======================
'  Stampa un record sullo schermo.
' ===========================================================

SUB StampaRecord(VarRecord AS UnitaInMag) STATIC
   PRINT "Numero del pezzo      : "; VarRecord.NumPezzo
   PRINT "Descrizione           : "; VarRecord.Descrizione
   PRINT USING "Prezzo unitario : Lit###,###"; _
      VarRecord.PrezzoUnitario
   PRINT "Quantità              : "; VarRecord.Quantita
END SUB

' ====================== TrovaRecord ======================
'  Utilizza una ricerca binaria per trovare il record
'  nell'indice.
' ===========================================================

FUNCTION TrovaRecord%(Pezzo$, VarRecord AS UnitaInMag) _
   STATIC
   SHARED Indice() AS TipoIndice, NumDeiRecord

   ' Imposta i limiti superiore e inferiore della ricerca:
   RecordSup = NumDeiRecord
   RecordInf = 1

   ' Ricerca finché il limite superiore non è minore del
   ' limite inferiore:
   DO UNTIL (RecordSup < RecordInf)

      ' Determina il punto intermedio:
      PuntoInt = (RecordSup + RecordInf) \ 2

      ' Prova se è il record desiderato (RTRIM$() toglie
      ' gli spazi vuoti dalla fine di una stringa fissa):
      Prova$ = RTRIM$(Indice(PuntoInt).NumPezzo)

      ' Se è la stringa giusta, esce dal ciclo:
      IF Prova$ = Pezzo$ THEN
         EXIT DO

      ' Altrimenti, se sta cercando un valore più alto,
      ' alza il limite inferiore:
```

```
      ELSEIF Pezzo$ > Prova$ THEN
         RecordInf = PuntoInt + 1

      ' Se no, abbassa il limite superiore:
      ELSE
         RecordSup = PuntoInt - 1
      END IF
   LOOP

   ' Se il pezzo è stato trovato, ottiene il record dal file
   ' usando un pointer all'indice e imposta TrovaRecord% a
   ' VERO:
   IF Prova$ = Pezzo$ THEN
      GET #1, Indice(PuntoInt).NumRecord, VarRecord
      TrovaRecord% = VERO

   ' Se non è stato trovato, imposta TrovaRecord% a FALSO:
   ELSE
      TrovaRecord% = FALSO
   END IF
END FUNCTION

' ===================== VisNumPezzo ======================
'  Stampa un elenco di tutti i numeri di pezzo nelle parte
'  alta dello schermo.
' ========================================================

SUB VisNumPezzo STATIC
   SHARED Indice() AS TipoIndice, NumDeiRecord

   CONST NUMCOL = 8, LARGCOL = 80 \ NUMCOL
   ' In cima allo schermo stampa l'elenco dei numeri di
   ' pezzo dei record nel file.  Questo menu viene stampato
   ' in colonne della stessa lunghezza (tranne forse
   ' l'ultima, che può essere più corta delle altre):
   LungCol = NumDeiRecord
   DO WHILE LungCol MOD NUMCOL
      LungCol = LungCol + 1
   LOOP
   LungCol = LungCol \ NUMCOL
   Colonna = 1
   NumRecord = 1
```

```
      DO UNTIL NumRecord > NumDeiRecord
         FOR Riga = 1 TO LungCol
            LOCATE Riga, Colonna
            PRINT Indice(NumRecord).NumPezzo
            NumRecord = NumRecord + 1
            IF NumRecord > NumDeiRecord THEN EXIT FOR
         NEXT Riga
         Colonna = Colonna + LARGCOL
      LOOP

      LOCATE LungCol + 1, 1
      PRINT STRING$(80, "_")        ' Stampa la linea divisoria.

      ' Fa scorrere le informazioni sui record sotto l'elenco
      ' dei numeri di pezzo, così i numeri non vengono
      ' cancellati:
      VIEW PRINT LungCol + 2 TO 24
   END SUB
```

Output

```
0235      0417      0583      8721
__________________________________________________________________
Digitare uno dei numeri di pezzo elencati in
alto (o U per uscire), e premere <INVIO>: 0417
Numero del pezzo: 0417
Descrizione: filtro olio
Prezzo unitario : Lit  6500
Quantità    : 12
________________________________________
Digitare uno dei numeri di pezzo elencati in
alto (o U per uscire), e premere <INVIO>: 8721
Numero del pezzo: 8721
Descrizione: pelle daino
Prezzo unitario : Lit 15000
Quantità    : 23
________________________________________
Digitare uno dei numeri di pezzo elencati in
alto (o U per uscire), e premere <INVIO>:
```

3.6.3 Emulatore di terminale (TERMINAL.BAS)

Il semplice programma che segue trasforma il computer in un terminale "stupido"; cioé lo fa funzionare esclusivamente come periferica per l'I/O su un modem. Questo programma si serve dell'istruzione **OPEN "COM1:"** e delle funzioni associate di I/O su periferica, per:

- Inviare al modem i caratteri che vengono digitati
- Visualizzare sullo schermo i caratteri inviati dal modem

Si noti che i caratteri digitati visualizzati sullo schermo vengono dapprima inviati al modem, e poi restituiti al computer attraverso il canale di comunicazione aperto. Questo si può verificare se si possiede un modem con segnali di rilevamento di ricezione (RD) e di invio (SD) – essi lampeggeranno in sequenza durante la digitazione.

Per impostare un numero telefonico, digitare dalla tastiera il comando Attention Dial Touch-Tone (ATDT), oppure Attention Dial Pulse (ATDP), (presumendo che si utilizzi un modem compatibile all'Hayes).

Ogni altro comando inviato al modem deve essere pure digitato dalla tastiera.

Istruzioni e funzioni utilizzate

Questo programma usa le seguenti funzioni ed istruzioni per la comunicazione con un modem attraverso la porta seriale del computer:

- **OPEN "COM1:"**
- **EOF**
- **INPUT$**
- **LOC**

Listato del programma

```
DEFINT A-Z

DECLARE SUB Filtro(StringaIn$)

COLOR 7, 1                          ' Imposta il colore dello
                                    ' schermo.
CLS
```

```
Uscita$ = CHR$(0) + CHR$(22)      ' Valore restituito da
                                  ' INKEY$ alla pressione di
                                  ' ALT+u.

' Posiziona il prompt all'ultima riga dello schermo e attiva
' il cursore:
LOCATE 24, 1, 1
PRINT STRING$(80, "_");
LOCATE 25, 1
PRINT TAB(30); "Premere ALT+u per uscire";

VIEW PRINT 1 TO 23                ' Stampa tra le righe 1
                                  ' e 23.

' Apre la porta di comunicazione (1200 baud, nessuna parità,
' dati a 8 bit, 1 bit di stop, buffer per l'input di 256
' byte):
OPEN "COM1:1200,N,8,1" FOR RANDOM AS #1 LEN = 256

DO                                ' Ciclo di comunicazioni
                                  ' principale.
   TastoInput$ = INKEY$           ' Controlla lo schermo.

   IF TastoInput$ = Uscita$ THEN  ' Esce dal ciclo se
      EXIT DO                     ' l'utente ha premuto
                                  ' ALT+u.
   ELSEIF TastoInput$ <> "" THEN  ' Se no, se l'utente ha
      PRINT #1, TastoInput$;      ' premuto  un tasto, lo
                                  ' invia al modem.
   END IF

   ' Controlla il modem. Se ci sono caratteri in attesa
   ' (EOF(1) è vero), li legge e li stampa sullo schermo:
   IF NOT EOF(1) THEN

      ' LOC(1) restituisce il numero dei caratteri in
      ' attesa:
      ModemInput$ = INPUT$(LOC(1), #1)

      Filtro ModemInput$          ' Filtra i caratteri di
      PRINT ModemInput$;          ' interlinea e di ritorno
   END IF                         ' unitario, poi stampa.
LOOP
```

```
CLOSE                                ' Chiude le
                                     ' communicazioni.
CLS
END

' ======================== FILTRO ==========================
'  Filtra i caratteri nella stringa di input.
' ==========================================================

SUB Filtro(StringaIn$) STATIC

   ' Cerca i caratteri di ritorno unitario e li sostituisce
   ' con CHR$(29) (la freccia SINISTRA):
   DO
      RitUn = INSTR(StringaIn$, CHR$(8))
      IF RitUn THEN
         MID$(StringaIn$, RitUn) = CHR$(29)
      END IF
   LOOP WHILE RitUn

   ' Cerca i caratteri di interlinea e li elimina:
   DO
      InterLin = INSTR(StringaIn$, CHR$(10))
      IF InterLin THEN
         StringaIn$ = LEFT$(StringaIn$, InterLin - 1) + _
            MID$(StringaIn$, InterLin + 1)
      END IF
   LOOP WHILE InterLin

END SUB
```

4 Manipolazione di stringhe

Il presente capitolo spiega come manipolare le sequenze di caratteri ASCII, note come stringhe. La manipolazione delle stringhe è indispensabile per elaborare file di testo, ordinare dati e modificare l'input di dati a stringa.

Dopo aver letto questo capitolo, l'utente sarà in grado di eseguire i seguenti processi di manipolazione delle stringhe:

- Dichiarare le stringhe a lunghezza fissa
- Confrontare le stringhe ed archiviarle in ordine alfabetico
- Cercare una stringa all'interno di un'altra
- Ottenere una parte di una stringa
- Eliminare gli spazi dall'inizio o la fine di una stringa
- Generare una stringa ripetendo un solo carattere
- Sostituire le lettere maiuscole di una stringa con lettere minuscole e viceversa
- Convertire le espressioni numeriche in rappresentazioni a stringa e viceversa

4.1 Definizione di stringa

Una stringa è una sequenza di caratteri contigui. Esempi di caratteri sono le lettere dell'alfabeto (a-z, e A-Z), i segni di punteggiatura come virgole (,) o punti interrogativi (?), ed altri simboli matematici e finanziari quali i segni più (+) e di percentuale (%).

In questo capitolo, il termine "stringa" si può riferire a:

- Una costante a stringa
- Una variabile a stringa
- Un'espressione a stringa

Le costanti a stringa vengono dichiarate in uno dei due modi seguenti:

- Racchiudendo una sequenza di caratteri tra doppie virgolette, come nella seguente istruzione **PRINT**:

  ```
  PRINT "Elaborazione del file in corso. Attendere..."
  ```

  Questa è detta "costante a stringa letterale".

- Assegnando a un nome il valore di una stringa letterale con un'istruzione **CONST**, come segue:

  ```
  ' Definisce la costante a stringa MESSAGGIO:
  CONST MESSAGGIO = "Lo sportello dell'unità è aperto."
  ```

  Questa è detta "costante a stringa simbolica".

Le variabili a stringa possono essere dichiarate in uno dei tre modi seguenti:

- Aggiungendo al nome della variabile il suffisso ($) del tipo a stringa:

  ```
  LINE INPUT "Nome e cognome: "; Buffer$
  ```

- Utilizzando l'istruzione **DEFSTR**:

  ```
  ' Tutte le variabili che iniziano per "B" rappresentano
  ' stringhe a meno che non terminino con uno dei suffissi
  ' propri di un tipo numerico (%, &, ! o #):
  DEFSTR B
  .
  .
  .
  Buffer = "Digitare il nome"      ' Buffer è una variabile
                                   ' a stringa.
  ```

- Dichiarando il nome di una stringa in un'istruzione **AS STRING**:

```
DIM Buffer1 AS STRING              ' Buffer1 è una stringa
                                   ' a lunghezza variabile.
DIM Buffer2 AS STRING * 10         ' Buffer2 è una stringa
                                   ' a lunghezza fissa
                                   ' lunga 10 byte.
```

Un'espressione a stringa è una combinazione di variabili, costanti e/o funzioni a stringa.

Naturalmente, i caratteri componenti le stringhe sono archiviati in memoria in maniera riconoscibile solo al computer: ciascun carattere viene tradotto in un numero noto come il codice ASCII del carattere. Per esempio, la lettera maiuscola "A" viene memorizzata come il numero decimale 65 (o esadecimale 41H). L'elenco completo dei codici ASCII è contenuto nell'appendice D, "Codici della tastiera e codici dei caratteri ASCII".

Per determinare il codice ASCII di un carattere, si può anche utilizzare la funzione **ASC** del BASIC; per esempio, `ASC("A")` restituisce il valore 65. La funzione inversa di **ASC** è la funzione **CHR$**. **CHR$** prende come argomento un codice ASCII e restituisce il carattere con quel codice; per esempio, l'istruzione `PRINT CHR$(64)` visualizza il carattere `@`.

4.2 Stringhe a lunghezza variabile e fissa

Nelle precedenti versioni del BASIC, le stringhe erano sempre a lunghezza variabile. Ora invece il BASIC supporta sia le stringhe a lunghezza variabile che quelle a lunghezza fissa.

4.2.1 Stringhe a lunghezza variabile

Le stringhe a lunghezza variabile sono "elastiche", cioè si espandono e si restringono per memorizzare un diverso numero di caratteri (da zero ad un massimo di 32767). Nell'esempio seguente la lunghezza della variabile `Temp$` varia secondo la dimensione di ciò che viene memorizzato in `Temp$`:

```
Temp$ = "1234567"

' La funzione LEN restituisce la lunghezza della stringa
' (quanti caratteri contiene):
PRINT LEN(Temp$)

Temp$ = "1234"
PRINT LEN(Temp$)
```

Output

```
 7
 4
```

4.2.2 Stringhe a lunghezza fissa

Le stringhe a lunghezza fissa vengono comunemente utilizzate come elementi di un record in un tipo di dati **TYPE...END TYPE** definito dall'utente. Esse si possono comunque dichiarare da sole nelle istruzioni **COMMON**, **DIM**, **REDIM**, **SHARED** o **STATIC**, come nell'istruzione seguente:

```
DIM Buffer AS STRING * 10
```

Come indica il loro nome, le stringhe a lunghezza fissa hanno una lunghezza costante, a prescindere dalla lunghezza della stringa in esse memorizzata. Ciò è evidente dall'output dell'esempio seguente:

```
DIM Nome AS STRING * 10
DIM Cognome AS STRING * 16

Nome = "Felix"
Cognome = "Mendelssohn-Bartholdy"

PRINT "123456789012345678901234567890"
PRINT Nome; Cognome
PRINT LEN(Nome)
PRINT LEN(Cognome)
```

Output

```
123456789012345678901234567890
Felix     Mendelssohn-Bart
 10
 16
```

Si noti che la lunghezza delle variabili a stringa `Nome` e `Cognome` non è stata modificata, come dimostrano i valori restituiti dalla funzione **LEN** (e i cinque spazi che seguono il nome di cinque lettere `Felix`).

L'output dell'esempio precedente mostra il modo in cui i valori assegnati alle variabili a lunghezza fissa vengono allineati a sinistra e, se necessario, troncati a destra. Non è necessario utilizzare la funzione **LSET** per allineare a sinistra i valori delle stringhe a lunghezza fissa, perché ciò avviene automaticamente.

Si deve invece utilizzare **RSET** se si vuole allineare a destra una stringa all'interno di una stringa a lunghezza fissa, come si vede qui:

```
DIM BufferNome AS STRING * 10
RSET BufferNome = "Felix"
PRINT "1234567890"
PRINT BufferNome
```

Output

```
1234567890
     Felix
```

4.3 Concatenazione di stringhe

Due stringhe possono essere unite tramite l'operatore più (+). La stringa che segue questo operatore viene aggiunta a quella che lo precede, come mostra l'esempio seguente:

```
A$ = "prima stringa"
B$ = "seconda stringa"
C$ = A$ + B$
PRINT C$
```

Output

```
prima stringaseconda stringa
```

Una unione di stringhe di questo tipo viene detta "concatenazione".

Si noti che, nell'esempio precedente, le due stringhe vengono combinate senza frapporre alcuno spazio. Se si vuole uno spazio all'interno del risultato, lo si può inserire in una delle stringhe `A$` o `B$` in questo modo:

```
B$ = " seconda stringa"          ' Con spazio iniziale in B$
```

Poiché nelle stringhe a lunghezza fissa i valori vengono allineati a sinistra, è possibile che risultino degli spazi vuoti quando esse vengono concatenate, come in questo esempio:

```
TYPE TipoNome
   Nome AS STRING * 10
   Cognome AS STRING * 16
END TYPE
```

```
DIM RecordNome AS TipoNome

' La costante "Paolo" è allineata a sinistra nella
' variabile RecordNome.Nome; ci sono cinque spazi vuoti:
RecordNome.Nome = "Paolo"
RecordNome.Cognome = "Uccello"

' Stampa una riga di numeri come confronto:
PRINT "123456789012345678901234567890"

NomeECognome$ = RecordNome.Nome + RecordNome.Cognome
PRINT NomeECognome$
```

Output

```
123456789012345678901234567890
Paolo     Uccello
```

La funzione **LTRIM$** restituisce una stringa senza gli eventuali spazi vuoti iniziali, e la funzione **RTRIM$** una stringa senza quelli finali. La stringa originale rimane inalterata. (Per ulteriori informazioni su queste funzioni, consultare la sezione 4.6, "Estrazione di parti di stringhe".)

4.4 Confronto tra stringhe

Le stringhe si possono confrontare mediante i seguenti operatori relazionali:

Operatore	*Significato*
<>	Non uguale
=	Uguale
<	Minore di
>	Maggiore di
<=	Minore o uguale a
>=	Maggiore o uguale a

Un carattere è maggiore di un altro se il suo valore ASCII è maggiore. Per esempio, il valore ASCII della lettera "B" è maggiore di quello della lettera "A", per cui l'espressione `"B" > "A"` è vera.

Quando si confrontano due stringhe, il BASIC utilizza i valori ASCII dei caratteri corrispondenti. Il primo carattere diverso tra le due stringhe determina l'ordine alfabetico. Per esempio, le stringhe `"doorman"` e `"doormats"` sono uguali fino al settimo carattere (`"n"` e `"t"`). Poiché il valore ASCII di `"n"` è minore del valore ASCII di `"t"`, l'espressione `"doorman" < "doormats"` è vera. Si noti che i valori ASCII delle lettere seguono l'ordine alfabetico dalla A alla Z e dalla a alla z, per cui si possono utilizzare gli operatori relazionali sopra elencati per sistemare le stringhe in ordine alfabetico. Le maiuscole hanno un valore ASCII minore delle minuscole: in un elenco ordinato di stringhe, per esempio, `"ASCII"` precederà `"ascii"`.

Se sono uguali i caratteri corrispondenti di due stringhe della stessa lunghezza, sono uguali anche le stringhe. Se però le due stringhe in questione non sono della stessa lunghezza, la stringa più lunga è maggiore dell'altra. Per esempio, `"abc" = "abc"` e `"aaaaaaa" > "aaa"`, sono entrambe espressioni vere.

Gli spazi vuoti iniziali e finali sono importanti quando si fa un confronto tra stringhe. Per esempio, la stringa `"  test"` è minore di `"test"`, perché uno spazio vuoto (`" "`) è minore di una `"t"`; viceversa, la stringa `"test  "` è maggiore della stringa `"test"`. Quando si confrontano stringhe a lunghezza fissa e a lunghezza variabile, ricordarsi che quelle a lunghezza fissa possono contenere spazi finali.

4.5 Ricerca di stringhe

Uno dei processi più comuni nella manipolazione delle stringhe è la ricerca di una stringa all'interno di un'altra. La funzione **INSTR** (*stringa1*, *stringa2*) permette di riconoscere se *stringa2* è contenuta o meno nella *stringa1*, restituendo la posizione del carattere di *stringa1* (se ne esiste uno) col quale comincia la corrispondenza, come si vede nell'esempio seguente:

```
Stringa1$ = "Una riga di testo di 33 caratteri"
Stringa2$ = "caratteri"

PRINT "         1         2         3         4"
PRINT "1234567890123456789012345678901234567890"
PRINT Stringa1$
PRINT Stringa2$
PRINT INSTR(Stringa1$, Stringa2$)
```

Output

```
         1         2         3         4
1234567890123456789012345678901234567890
Una riga di testo di 33 caratteri
caratteri
 25
```

Se non viene rilevata alcuna corrispondenza (cioè *stringa2* non è una sottostringa di *stringa1*), **INSTR** restituisce il valore 0. Una variazione della sintassi precedente, **INSTR** (*posizione*, *stringa1*, *stringa2*), permette di specificare il punto in cui si vuole cominciare la ricerca in *stringa1*. Apportando la seguente modifica all'esempio precedente, varia anche l'output:

```
' Inizia a cercare la corrispondenza al carattere
' no. 30 di Stringa1$:
PRINT INSTR(30, Stringa1$, Stringa2$)
```

Output

```
         1         2         3         4
1234567890123456789012345678901234567890
Una riga di testo di 33 caratteri
caratteri
 0
```

In altri termini, la stringa `"caratteri"` non appare dopo il trentesimo carattere di `Stringa1$`.

La versione **INSTR** (*posizione*, *stringa1*, *stringa2*) di questa funzione è utile per il rilevamento di ogni istanza di *stringa2* in *stringa1*, invece che della prima soltanto, come mostra l'esempio seguente:

```
Stringa1$ = "volli, sempre volli, fortissimamente volli"
Stringa2$ = "volli"

Inizio = 1
NumCorrisp = 0
```

```
DO
   Corrisp = INSTR(Inizio, Stringa1$, Stringa2$)
   IF Corrisp > 0 THEN
      PRINT TAB(Corrisp); Stringa2$
      Inizio = Corrisp + 1
      NumCorrisp = NumCorrisp + 1
   END IF
LOOP WHILE Corrisp

PRINT "Il numero delle corrispondenze è"; NumCorrisp
```

Output

```
volli, sempre volli, fortissimamente volli
volli
              volli
                                       volli

Il numero delle corrispondenze è 3
```

4.6 Estrazione di parti di stringa

La sezione 4.3, "Concatenazione di stringhe", spiega come unire le stringhe (concatenarle) utilizzando l'operatore +. Nel BASIC sono disponibili diverse funzioni che frazionano le stringhe, restituendone frammenti dalla sinistra, dalla destra, oppure dall'interno.

4.6.1 Estrazione di caratteri dal lato sinistro di una stringa

Le funzioni **LEFT$** e **RTRIM$** estraggono caratteri dal lato sinistro di una stringa. La funzione **LEFT$** (*stringa*, *numero*) estrae il *numero* specificato di caratteri dal lato sinistro della *stringa*. Per esempio:

```
Test$ = "Apertamente"

' Stampa i 6 caratteri più a sinistra in Test$:
PRINT LEFT$(Test$, 6)
```

Output

```
Aperta
```

Si noti che **LEFT$**, così come le altre funzioni descritte in questo capitolo, non modifica la stringa originale `Test$`, ma restituisce semplicemente un'altra stringa, una copia di parte della stringa `Test$`.

La funzione **RTRIM$** restituisce la parte sinistra di una stringa dopo aver eliminato qualunque spazio vuoto che si trovi alla fine di essa. Per esempio, si confronti l'output delle due istruzioni **PRINT** nell'esempio successivo:

```
PRINT "stringa giustificata a sinistra    "; "parola"
PRINT RTRIM$("stringa giustificata a sinistra    "); _
   "parola"
```

Output

```
stringa giustificata a sinistra    parola
stringa giustificata a sinistraparola
```

La funzione **RTRIM$** è utile per confrontare stringhe a lunghezza fissa con stringhe a lunghezza variabile, come nell'esempio seguente:

```
DIM NomeRecord AS STRING * 10
NomeRecord = "Paolo"

NomeTest$ = "Paolo"

' Usa RTRIM$ per saltare gli spazi vuoti alla destra della
' stringa a lunghezza fissa NomeRecord, poi la confronta
' con la stringa a lunghezza variabile NomeTest$:
IF RTRIM$(NomeRecord) = NomeTest$ THEN
   PRINT "Sono uguali"
ELSE
   PRINT "Non sono uguali"
END IF
```

Output

```
Sono uguali
```

4.6.2 Estrazione di caratteri dal lato destro di una stringa

Le funzioni **RIGHT$** e **LTRIM$** estraggono caratteri dal lato destro di una stringa. La funzione **RIGHT$** (*stringa*, *numero*) estrae il *numero* specificato di caratteri dal lato destro della stringa. Per esempio:

```
Test$ = "1234567890"

' Stampa i 5 caratteri più a destra in Test$:
PRINT RIGHT$(Test$, 5)
```

Output

```
67890
```

La funzione **LTRIM$** restituisce la parte destra di una stringa dopo aver eliminato gli eventuali spazi vuoti all'inizio di essa. Si confronti, per esempio, l'output delle due istruzioni **PRINT** successive:

```
PRINT "parola"; "    stringa giustificata a destra"
PRINT "parola"; LTRIM$("    stringa giustificata a destra")
```

Output

```
parola    stringa giustificata a destra
parolastringa giustificata a destra
```

4.6.3 Estrazione di caratteri da qualunque parte di una stringa

Per estrarre un numero qualsiasi di caratteri da un punto qualsiasi di una stringa viene utilizzata la funzione **MID$**. La funzione **MID$** (*stringa*, *inizio*, *numero*) estrae dalla *stringa* il *numero* specificato di caratteri, cominciando dal carattere in posizione *inizio*. Per esempio l'istruzione:

```
MID$("**ed è subito sera**", 15, 4)
```

inizia al quindicesimo carattere (la seconda `s`) della stringa

```
**ed è subito sera**
```

e restituisce questo carattere più i successivi tre (`sera`).

L'esempio seguente spiega il modo in cui utilizzare la funzione **MID$** per scorrere una riga di testo un carattere alla volta:

```
.
.
.
' Ottiene il numero di caratteri di una stringa di testo:
Lung = LEN(StringaTesto$)

FOR I = 1 TO Lung

   ' Legge il primo carattere, poi il secondo, il terzo
   ' e così via, fino alla fine della stringa:
   Car$ = MID$(StringaTesto$, I, 1)

   ' Elabora il carattere:
   .
   .
   .
NEXT
```

4.7 Generazione di stringhe

Il modo più semplice per creare una stringa composta da un carattere ripetuto più volte è quello di utilizzare la funzione intrinseca **STRING$**. La funzione **STRING$** (*numero*, *stringa*) produce una nuova stringa con lo specificato *numero* di caratteri, ciascuno dei quali è il primo carattere dell'argomento *stringa*. Per esempio, l'istruzione

```
Tampone$ = STRING$(27, "*")
```

genera una stringa di 27 asterischi. La forma alternativa di questa funzione, **STRING$** (*numero*, *codice*), viene utilizzata per quei caratteri che non possono essere prodotti mediante digitazione, quali quelli i cui valori ASCII sono maggiori di 127. Questa forma crea una stringa con il numero precisato di caratteri, ciascuno dei quali ha il codice ASCII specificato dall'argomento *codice*, come nell'esempio seguente:

```
' Stampa una stringa di 10 caratteri "@" (64 è il codice
' ASCII del carattere @)
PRINT STRING$(10, 64)
```

Output

```
@@@@@@@@@@
```

La funzione **SPACE$** (*numero*) genera una stringa composta dal precisato *numero* di spazi vuoti.

4.8 Sostituzione delle minuscole con maiuscole e viceversa

E' possibile trasformare le lettere maiuscole di una stringa in minuscole o viceversa. Questa conversione può essere utile per la ricerca di una particolare stringa all'interno di un file esteso, quando non abbiano rilevanza le maiuscole (per esempio, se `guida`, `GUIDA` e `Guida` siano da considerare tutti uguali). Queste funzioni sono utili anche quando non si è sicuri se un utente digiterà del testo con lettere maiuscole o minuscole.

Le funzioni **UCASE$** e **LCASE$** operano le seguenti conversioni in una stringa:

- **UCASE$** restituisce una copia della stringa ad essa passata, con le lettere minuscole trasformate in maiuscole.
- **LCASE$** restituisce una copia della stringa ad essa passata, trasformando le lettere maiuscole in minuscole.

Esempio

```
Saggio$ = "* L'appendice ASCII: una tabella di codici _
   utili *"
PRINT UCASE$(Saggio$)
PRINT LCASE$(Saggio$)
PRINT Saggio$
```

Output

```
* L'APPENDICE ASCII: UNA TABELLA DI CODICI UTILI *
* l'appendice ascii: una tabella di codici utili *
* L'appendice ASCII: una tabella di codici utili *
```

Le lettere che sono già maiuscole, così come i caratteri che non sono lettere, non vengono modificate da **UCASE$**; analogamente, le lettere minuscole e i caratteri che non sono lettere non vengono modificati da **LCASE$**.

4.9 Stringhe e numeri

Il BASIC non ammette che una stringa sia assegnata ad una variabile numerica, né che un'espressione numerica sia assegnata ad una variabile a stringa. Per esempio, entrambe queste istruzioni danno luogo al messaggio di errore `Mancata corrispondenza dei tipi`:

```
BufferTemp$ = 45
Conteggio% = "45"
```

Usare invece la funzione **STR$** per ottenere la rappresentazione a stringa di un numero, e la funzione **VAL** per ottenere la rappresentazione numerica di una stringa:

```
' Le seguenti istruzioni sono valide:
BufferTemp$ = STR$(45)
Conteggio% = VAL("45")
```

Si noti che **STR$** include lo spazio vuoto iniziale che il BASIC stampa per i numeri positivi, come mostra questo breve esempio:

```
FOR I = 0 TO 9
   PRINT STR$(I);
NEXT
```

Output

```
0 1 2 3 4 5 6 7 8 9
```

Se si vuole eliminare questo spazio, si può utilizzare la funzione **LTRIM$**, come nell'esempio seguente:

```
FOR I = 0 TO 9
   PRINT LTRIM$(STR$(I));
NEXT
```

Output

```
0123456789
```

Un altro modo per formattare un output numerico consiste nell'utilizzare l'istruzione **PRINT USING** (consultare la sezione 3.1, "Visualizzazione del testo sullo schermo").

4.10 Modifica di stringhe

Tutte le funzioni fin qui descritte lasciano inalterati i propri argomenti a stringa: ogni modifica viene fatta solamente su di una copia.

L'istruzione **MID$** invece (a differenza della funzione **MID$** trattata nella sezione 4.6.3, "Estrazione di caratteri da qualunque parte di una stringa") modifica il proprio argomento inserendo un'altra stringa al suo interno. Tale stringa rimpiazza in parte o del tutto la stringa originale senza cambiarne la lunghezza, come è illustrato nell'esempio seguente:

```
Temp$ = "1234567890"
PRINT Temp$

' Sostituisce "1" con "a":
MID$(Temp$,1) = "a"

' Sostituisce "90" con "il":
MID$(Temp$,9) = "ilm"
PRINT Temp$
```

Output

```
1234567890
a2345678il
```

4.11 Programma esempio: trasformazione di una stringa in un numero (STRINNUM.BAS)

Il programma seguente legge il numero immesso come una stringa, ne elimina i caratteri non numerici (quali le virgole), quindi con la funzione **VAL** la converte in un numero.

Istruzioni e funzioni utilizzate

Questo programma presenta le seguenti funzioni per la gestione di stringhe trattate in questo capitolo:

- **INSTR**
- **LEN**
- **MID$**
- **VAL**

Listato del programma

```
DECLARE FUNCTION Filtro$(Testo$, StringaFiltro$)

' Legge una riga:
LINE INPUT "Digitare un numero con virgole: ", A$

' Conserva solo i caratteri numerici (0123456789.-) nella
' stringa digitata:
NumPuro$ = Filtro$(A$, "0123456789.-")

' Converte la stringa in numero:
PRINT "Il valore numerico è "; VAL(NumPuro$)
END

' ======================== FILTRO ==========================
'  Elimina i caratteri non desiderati da una stringa
'  confrontandoli con una stringa filtro composta dai
'  caratteri numerici accettabili.
' ==========================================================

FUNCTION Filtro$(Testo$, StringaFiltro$) STATIC
   Temp$ = ""
   LungTesto = LEN(Testo$)

   FOR I = 1 TO LungTesto      ' Isola ogni carattere della
                               ' stringa.
      C$ = MID$(Testo$, I, 1)

      ' Se il carattere è nella stringa filtro, lo salva:
      IF INSTR(StringaFiltro$, C$) <> 0 THEN
         Temp$ = Temp$ + C$
      END IF
   NEXT I

   Filtro$ = Temp$
END FUNCTION
```

5 Grafica

Questo capitolo spiega come utilizzare le istruzioni e le funzioni grafiche del BASIC per creare sullo schermo un'ampia varietà di figure, colori e motivi. Con le funzioni grafiche un programma acquista una dimensione visiva, sia esso un gioco, uno strumento educativo, un'applicazione scientifica o un package finanziario. A lettura ultimata si sapranno eseguire le seguenti funzioni grafiche:

- Utilizzare le coordinate fisiche dello schermo per individuare singoli pixel, accenderli, spegnerli e modificarne i colori
- Disegnare linee
- Disegnare e colorare forme semplici, quali cerchi, ovali e riquadri
- Controllare l'area dello schermo per l'output grafico, utilizzando finestre
- Regolare le coordinate utilizzate per individuare i pixel, ridefinendo le coordinate dello schermo
- Utilizzare il colore nell'output grafico
- Creare motivi e utilizzarli per colorare figure chiuse
- Copiare immagini da un punto dello schermo a un altro
- Animare l'output grafico

La sezione successiva presenta del materiale necessario per eseguire gli esempi grafici presentati in questo capitolo. E' necessario pertanto leggerla attentamente.

5.1 Requisiti per i programmi grafici

Per eseguire gli esempi grafici illustrati in questo capitolo, il computer deve avere capacità grafiche: deve essere fornito di adattatore grafico quale una scheda CGA, EGA o VGA. E' necessario inoltre un monitor (monocromatico o a colori) che supporti la grafica basata sui pixel.

Condizione necessaria ai programmi seguenti è anche che lo schermo si trovi in una delle "modalità" che supporta l'output grafico. (La modalità schermo controlla la chiarezza delle immagini grafiche, il numero dei colori disponibili, e la parte di memoria video da visualizzare.) Per selezionare una modalità di output grafico, inserire un'istruzione **SCREEN** *modalità* nel programma prima di qualsiasi altra istruzione o funzione grafica descritta in questo capitolo.

Con l'istruzione **SCREEN**, la *modalità* può essere 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, o 13, a seconda della combinazione monitor/adattatore installata nel computer.

Se non si è sicuri che l'utilizzatore del programma abbia o meno un hardware che supporti la grafica, può risultare utile effettuare il seguente test:

```
CONST FALSO = 0, VERO = NOT FALSO

' Prova se l'utente dispone dell'hardware necessario per
' la grafica a colori:
ON ERROR GOTO Messaggio        ' Attiva gestione errori.
SCREEN 1                       ' Prova la modalità grafica 1.
ON ERROR GOTO 0                ' Disattiva gestione errori.
IF GraficaNo THEN END          ' Termina se manca l'hardware
.                              ' per la grafica.
.
.
END

' Routine gestione errori:
Messaggio:
   PRINT "Questo programma richiede capacità grafica."
   GraficaNo = VERO
   RESUME NEXT
```

Se l'utente possiede solo un monitor monocromatico sprovvisto di adattatore grafico, l'istruzione **SCREEN** produrrà un errore che farà saltare alla routine di gestione degli errori `Messaggio`. (Per ulteriori informazioni relative al rilevamento di errori consultare il capitolo 6, "Gestione degli errori e degli eventi".)

5.2 Pixel e coordinate di schermo

Le forme e le figure vengono rappresentate su un monitor con singoli punti luminosi noti come "pixel" (o anche "pel"). Il BASIC disegna e colora sullo schermo accendendo o spegnendo questi pixel e modificandone il colore.

Uno schermo tipico è costituito da una griglia di pixel. Il numero esatto di pixel di questa griglia dipende dall'hardware installato e dalla modalità schermo selezionata con l'istruzione **SCREEN**. Maggiore è il numero dei pixel, più nitido è l'output grafico.

Per esempio, un'istruzione **SCREEN 1** fornisce una risoluzione di 320 pixel in orizzontale e 200 in verticale (320 x 200 pixel), mentre un'istruzione **SCREEN 2** fornisce una risoluzione di 640 x 200 pixel; con quest'ultima si otterrà quindi una più alta risoluzione nelle figure, che appariranno più nitide.

A seconda delle possibilità grafiche del sistema, si possono utilizzare altre modalità schermo che supportano risoluzioni ancora più alte (e allo stesso tempo regolano altre caratteristiche dello schermo). Per ulteriori informazioni consultare il Consulente QB.

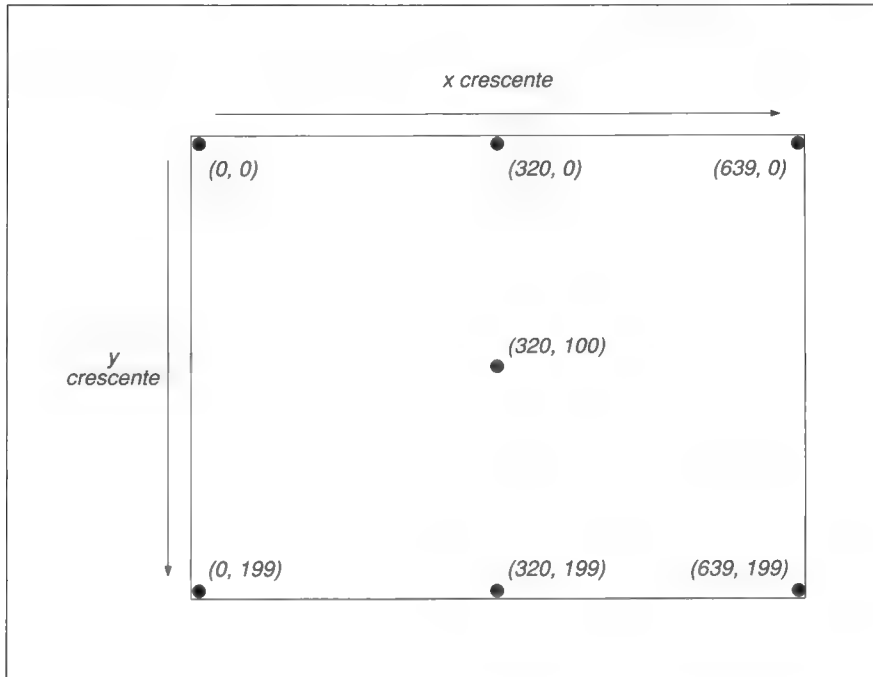
Se lo schermo si trova in una delle modalità grafiche, è possibile individuare ciascun pixel per mezzo di una coppia di coordinate. Il primo numero di ogni coppia di coordinate indica la distanza in pixel dal lato sinistro dello schermo, mentre il secondo indica la distanza in pixel dal lato superiore. Per esempio, in modalità schermo 2 il punto all'estremo angolo in alto a sinistra ha coordinate (0, 0), quello al centro dello schermo ha coordinate (320, 100), quello all'estremo angolo in basso a destra ha coordinate (639, 199), come si vede nell'illustrazione 5.1.

Il BASIC utilizza queste coordinate di schermo per determinare il punto in cui visualizzare i grafici (per esempio, per localizzare i punti estremi di una linea o il centro di un cerchio), come è illustrato nella sezione 5.3, "Disegno di forme fondamentali".

Le coordinate grafiche si differenziano da quelle di modalità testo determinate da un'istruzione **LOCATE**. Innanzi tutto, **LOCATE** non è altrettanto preciso: le coordinate grafiche individuano con precisione singoli pixel sullo schermo, mentre le coordinate utilizzate da **LOCATE** indicano la posizione di un intero carattere. Inoltre, le coordinate in modalità testo vengono fornite nell'ordine "riga, colonna", in questo modo:

```
' Si porta alla 13ma riga, 15ma colonna, e stampa
' il messaggio indicato:
LOCATE 13, 15
PRINT "Il presente appare al centro dello schermo."
```

Per le coordinate grafiche si verifica l'inverso: queste, infatti, vengono fornite nell'ordine "colonna, riga". Un'istruzione **LOCATE** non influisce sulla posizione degli output grafici sullo schermo.

Illustrazione 5.1 Coordinate di pixel selezionati in modalità schermo 2

5.3 Disegno di forme fondamentali: punti, righe, riquadri e cerchi

Si possono passare alle istruzioni grafiche del BASIC i valori delle coordinate per produrre una varietà di figure semplici, come è mostrato nelle sezioni 5.3.1–5.3.2.

5.3.1 Tratteggio di punti con PSET e PRESET

La semplice attivazione e disattivazione dei singoli pixel costituisce il controllo basilare sull'output grafico. Nel BASIC questo controllo si ottiene con le istruzioni **PSET** e **PRESET**. L'istruzione **PSET** (*x*, *y*) dà al pixel in posizione (*x*, *y*) il colore di primo piano, e **PRESET** (*x*, *y*) il colore di sfondo.

Su monitor monocromatici, il colore di primo piano è quello utilizzato per visualizzare il testo, ed è di solito bianco, ambra o verde chiaro; il colore di sfondo invece è di solito nero o verde scuro. Si può scegliere un altro colore per **PSET** o **PRESET** aggiungendo l'argomento opzionale *colore*. La sintassi è la seguente:

PSET (*x*, *y*), *colore*

oppure

PRESET (*x*, *y*), *colore*

Per ulteriori informazioni relative alla scelta dei colori, consultare la sezione 5.7.

Poiché per predefinizione **PSET** utilizza il colore corrente di primo piano e **PRESET** quello corrente di sfondo, **PRESET** senza l'argomento *colore* cancella un punto disegnato con **PSET**, come è illustrato nell'esempio seguente:

```
SCREEN 2                ' Risoluzione 640 x 200

PRINT "Premere un tasto per terminare."

DO

   ' Traccia una linea orizzontale da sinistra a destra:
   FOR X = 0 TO 640
      PSET (X, 100)
   NEXT

   ' Cancella la linea da destra a sinistra:
   FOR X = 640 TO 0 STEP -1
      PRESET (X, 100)
   NEXT

LOOP UNTIL INKEY$ <> ""
END
```

Sebbene sia possibile disegnare qualunque figura manipolando singoli pixel con l'istruzione **PSET**, la visualizzazione dell'output viene rallentata quando l'illustrazione è composta da numerosi pixel. Il BASIC dispone di diverse istruzioni per velocizzare la rappresentazione di semplici figure quali linee, riquadri ed ellissi, come spiegato nelle sezioni 5.3.2 e 5.4.1–5.4.4.

5.3.2 Disegno di linee e riquadri con LINE

Con **PSET** e **PRESET** si specifica una sola coppia di coordinate riferendosi ad un singolo punto sullo schermo. Con **LINE**, si specificano due coppie, una per ciascuna estremità del segmento. La forma più semplice dell'istruzione **LINE** è la seguente:

LINE (*x1*, *y1*)–(*x2*, *y2*)

in cui (*x1*, *y1*) sono le coordinate di un'estremità di un segmento e (*x2*, *y2*) sono le coordinate dell'altra. Per esempio, l'istruzione seguente disegna un segmento di retta dal pixel con coordinate (10, 10) a quello con coordinate (150, 200):

```
SCREEN 1
LINE (10, 10)-(150, 200)
```

Si noti che il BASIC non prende in considerazione l'ordine delle coppie di coordinate: una linea da (*x1*, *y1*) a (*x2*, *y2*) è uguale a quella da (*x2*, *y2*) a (*x1*, *y1*). Quindi si può scrivere la precedente istruzione nel seguente modo:

```
SCREEN 1
LINE (150, 200)-(10, 10)
```

L'inversione dell'ordine delle coordinate potrebbe però avere un effetto nelle istruzioni grafiche seguenti, come viene spiegato nella prossima sezione.

5.3.2.1 Utilizzo dell'opzione STEP

Fino a questo momento, le coordinate di schermo sono state presentate come valori assoluti che misurano la distanza orizzontale e verticale dall'angolo superiore sinistro dello schermo, che ha coordinate (0, 0). Tuttavia, utilizzando l'opzione **STEP** nelle seguenti istruzioni grafiche, le coordinate successive all'istruzione **STEP** diventano relative all'ultimo punto tracciato sullo schermo:

CIRCLE	**GET**
LINE	**PAINT**
PRESET	**PSET**
PUT	

Se si pensa alle immagini sullo schermo come se fossero tracciate da un sottile pennello dell'esatta dimensione di un pixel, il punto in cui questo pennello o "cursore grafico" si ferma, dopo aver terminato di tracciare un'immagine, costituisce la sua posizione.

Per esempio, le istruzioni seguenti lasciano il cursore grafico nel punto in cui appare il pixel con coordinate (100, 150):

```
SCREEN 2
LINE (10, 10)-(100, 150)
```

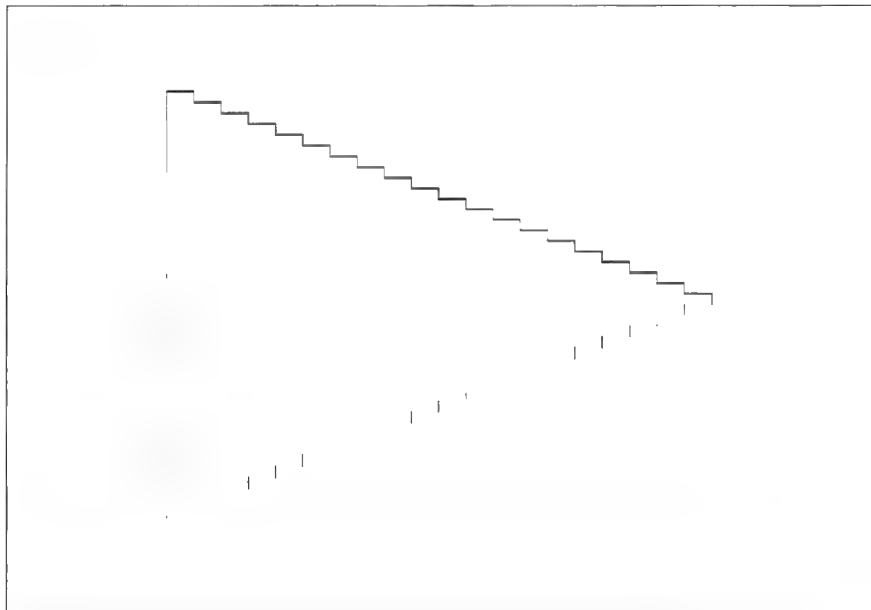
Se nel programma la istruzione grafica successiva è:

```
PSET STEP (20, 20)
```

allora il punto tracciato da **PSET** non sarà nel quadrante superiore sinistro dello schermo. Con l'opzione **STEP**, infatti, le coordinate (20, 20) diventano relative all'ultimo punto riferito, che ha coordinate (100, 150). Perciò le coordinate assolute del punto saranno (100 + 20, 150 + 20) ovvero (120, 170).

Nell'esempio illustrato sopra, l'ultimo punto riferito viene determinato dalla precedente istruzione grafica. Si può anche stabilire un riferimento a un punto nella stessa istruzione, come mostra l'esempio seguente:

```
' Imposta una risoluzione di 640 x 200 pixel, e il punto di
' riferimento al centro dello schermo:
SCREEN 2

' Traccia una linea dall'angolo in basso a sinistra
' all'angolo in alto a sinistra:
LINE STEP(-310, 100) -STEP(0, -200)

' Disegna le "scalette" dall'angolo in alto a sinistra
' al lato destro dello schermo:
FOR I% = 1 TO 20
LINE -STEP(30, 0)             ' Traccia la linea orizzontale.
LINE -STEP(0, 5)              ' Traccia la linea verticale.
NEXT

' Disegna i segmenti verticali individuali dal lato destro
' all'angolo in basso a sinistra:
FOR I% = 1 TO 20
   LINE STEP(-30, 0) -STEP(0, 5)
NEXT

DO: LOOP WHILE INKEY$ = ""    ' Aspetta la pressione di
                              ' un tasto.
```

Output

Nota Si noti il ciclo **DO** vuoto alla fine dell'ultimo programma. Se si sta eseguendo un programma autonomo BASIC compilato che genera dell'output grafico, questo ha bisogno di un meccanismo, quale appunto il ciclo vuoto sopra illustrato, per mantenere l'output sullo schermo alla fine, altrimenti l'output svanisce prima che l'utente abbia il tempo di osservarlo.

5.3.2.2 Disegno di riquadri

Si è visto che utilizzando le forme dell'istruzione **LINE**, è abbastanza semplice scrivere un programma breve che unisce quattro linee rette per formare un riquadro, come viene ora dimostrato:

```
SCREEN 1                ' Risoluzione 320 x 200 pixel

' Disegna un riquadro di 120 pixel per lato:
LINE (50, 50)-(170, 50)
LINE -STEP(0, 120)
LINE -STEP(-120, 0)
LINE -STEP(0, -120)
```

Tuttavia, il BASIC fornisce un metodo ancora più semplice per tracciare un riquadro, cioè utilizzando una sola istruzione **LINE** con l'opzione **B** (per box = riquadro). L'opzione **B** viene presentata nel prossimo esempio, che genera lo stesso output delle quattro istruzioni **LINE** del programma precedente:

```
SCREEN 1                ' Risoluzione 320 x 200 pixel

' Disegna un riquadro con coordinate (50, 50) per
' l'angolo superiore sinistro, e (170, 170) per
' quello inferiore destro:
LINE (50, 50)-(170, 170), , B
```

Quando si aggiunge l'opzione **B**, l'istruzione **LINE** non unisce con una retta i due punti dati; disegna invece un rettangolo i cui angoli opposti (quello superiore sinistro e quello inferiore destro) si trovano ai due punti dati.

Nell'ultimo esempio due virgole precedono l'opzione **B**. La prima funge da segnaposto per un'opzione non utilizzata (l'argomento *colore*), che permette di scegliere il colore di una linea o dei lati di un riquadro. (Per ulteriori informazioni relative all'utilizzazione del colore, consultare la sezione 5.7.)

Come è stato detto prima, poiché non ha alcuna importanza l'ordine in cui vengono sistemate le coppie di coordinate, il rettangolo dell'esempio precedente può essere disegnato anche con questa istruzione:

```
LINE (170, 170)-(50, 50), , B
```

Aggiungendo l'opzione **F** (per la colorazione: fill = riempi) all'opzione **B**, si assegna all'interno del riquadro lo stesso colore utilizzato per il bordo. Con un monitor monocromatico, questo colore corrisponderà al colore di primo piano utilizzato per il testo visualizzato. Se l'hardware ha la possibilità di utilizzare altri colori, si può cambiare questo colore mediante l'argomento opzionale *colore* (si consulti la sezione 5.7.1,"Scelta di un colore per output grafico").

La sintassi fin qui presentata per disegnare un riquadro è la sintassi generale utilizzata nel BASIC per definire una regione grafica rettangolare, che appare anche nelle istruzioni **GET** e **VIEW**:

{GET | LINE | VIEW} **(***x1*, *y1***)–(***x2*, *y2***), ...**

Qui, (*x1*, *y1*) e (*x2*, *y2*) sono le coordinate degli angoli diagonalmente opposti del rettangolo (superiore sinistro e inferiore destro). (Per ulteriori informazioni su **VIEW** e su **GET** e **PUT**, si consultino le sezioni 5.5, "Definizione di una finestra grafica", e 5.10, "Tecniche fondamentali di animazione".)

5.3.2.3 Disegno di linee punteggiate

La sezione precedente spiega come utilizzare l'istruzione **LINE** per disegnare linee continue ed utilizzarle nei rettangoli, senza omettere alcun pixel. Tuttavia, utilizzando un'altra opzione con **LINE**, si possono invece disegnare linee tratteggiate o punteggiate. Questa procedura è nota come "line styling". Quella che segue è la sintassi per disegnare una singola linea tratteggiata dal punto (*x1*, *y1*) al punto (*x2*, *y2*) utilizzando il colore corrente di primo piano:

LINE (*x1*, *y1*)-(*x2*, *y2*), , [**B**], *stile*

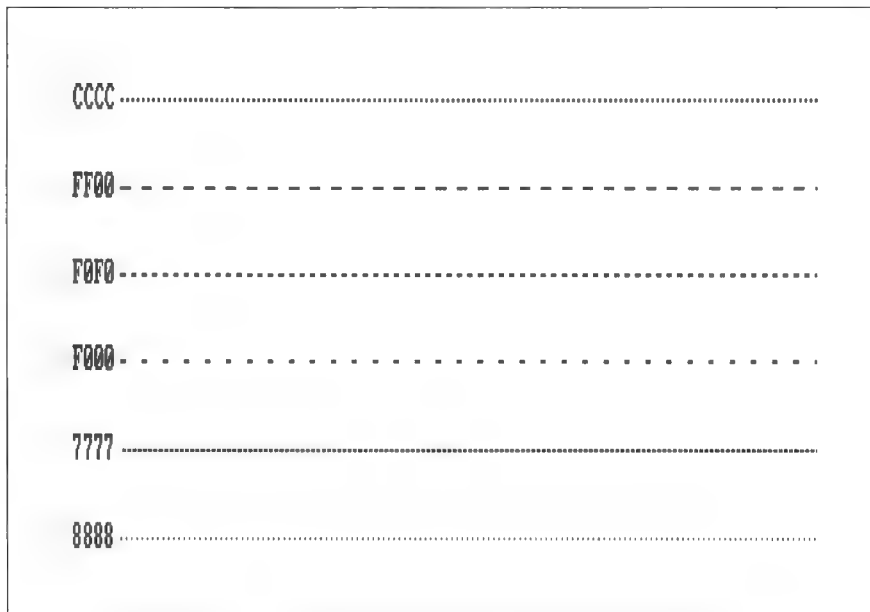
Qui, *stile* è un intero decimale o esadecimale a 16 bit. L'istruzione **LINE** utilizza la rappresentazione binaria dell'argomento *stile* per creare trattini e spazi: il bit 1 significa "accendere il pixel", ed il bit 0 significa "spegnere il pixel". Per esempio l'intero esadecimale &HCCCC (uguale all'intero binario 1100110011001100) utilizzato come argomento *stile* fa tracciare una linea in cui due pixel accesi si alternano con due spenti.

Esempio

Nell'esempio seguente, le diverse linee tratteggiate sono determinate dai diversi valori dell'argomento *stile*:

```
SCREEN 2                    ' Risoluzione di 640 X 200 pixel

' Dati per gli stili:
DATA &HCCCC, &HFF00, &HF0F0
DATA &HF000, &H7777, &H8888
Riga% = 4
Colonna% = 4
XSin% = 60
XDes% = 600
Y% = 28

FOR I% = 1 TO 6
   READ Stile%
   LOCATE Riga%, Colonna%
   PRINT HEX$(Stile%)
   LINE (XSin%, Y%)-(XDes%, Y%), , , Stile%
   Riga% = Riga% + 3
   Y% = Y% + 24
NEXT
```

Output

```
CCCC ............................................................
FF00 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
F0F0 ............................................................
F000 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
7777 ------------------------------------------------------------
8888 ............................................................
```

5.4 Disegno di cerchi ed ellissi con CIRCLE

L'istruzione **CIRCLE** disegna un numero svariato di forme circolari o ellittiche (ovali). Inoltre, disegna archi (segmenti di cerchi) e spicchi di torta. In modalità grafica, si possono generare quasi tutti i tipi di curva, per mezzo di variazioni di **CIRCLE**.

5.4.1 Disegno di cerchi

Per disegnare un cerchio è sufficiente conoscere il suo centro ed il suo raggio (la distanza dal centro a un qualunque punto del cerchio). Avvalendosi di queste informazioni e di una mano ferma (o, se non si è Giotto, utilizzando un compasso), si può disegnare un discreto cerchio.

Anche il BASIC, per disegnare un cerchio, ha bisogno delle stesse indicazioni. La forma più semplice della sintassi di **CIRCLE** è:

CIRCLE [STEP] (*x*, *y*), *raggio*

dove *x*, *y* sono le coordinate del centro, e *raggio* è il raggio del cerchio.

Nell'esempio seguente viene disegnato un cerchio con centro (200, 100) e raggio 75:

```
SCREEN 2
CIRCLE (200, 100), 75
```

Utilizzando l'opzione **STEP** all'interno della stessa sintassi, le coordinate diventano relative al centro dello schermo piuttosto che all'angolo superiore sinistro. Ne deriva:

```
SCREEN 2             ' Usa il centro dello schermo (320, 100)
                     ' come punto di riferimento delle
                     ' coordinate:
CIRCLE STEP(-120, 0), 75
```

5.4.2 Disegno di ellissi

L' istruzione **CIRCLE** regola automaticamente il "rapporto delle dimensioni" (aspetto) per accertarsi che i cerchi appaiano rotondi sullo schermo e non appiattiti. Tuttavia, potrebbe risultare necessario regolare il rapporto delle dimensioni per una buona visualizzazione dei cerchi sul monitor, o modificarlo per disegnare figure ovali, dette ellissi. In ciascuno dei casi, la sintassi è:

CIRCLE [STEP] (*x*, *y*), *raggio*, , , , *aspetto*

dove *aspetto* è un numero reale positivo. (Per ulteriori informazioni relative al rapporto delle dimensioni e al modo in cui calcolarlo in diverse modalità schermo, consultare la sezione 5.4.5.)

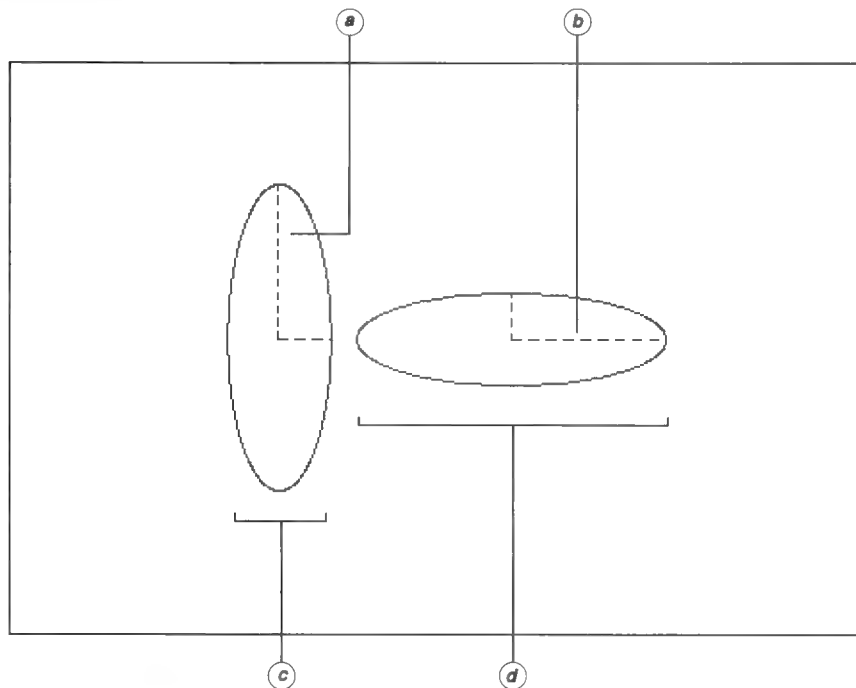
Le virgole extra tra *raggio* e *aspetto* sono segnaposti per altre opzioni che indicano all'istruzione **CIRCLE** quale colore utilizzare (se si possiede un sistema monitor / adattatore a colori e si utilizza una delle modalità schermo che supporta il colore), oppure se disegnare un arco o uno spicchio. (Per ulteriori informazioni relative a queste opzioni, consultare la sezione 5.4.3, "Disegno di archi", e 5.7.1, "Scelta di un colore per output grafico".)

Poiché l'argomento *aspetto* determina la proporzione delle dimensioni verticali rispetto a quelle orizzontali, grandi valori di aspetto producono delle ellissi che si estendono lungo l'asse orizzontale. Valori piccoli ne producono altre che si estendono invece lungo quello verticale. Poiché un'ellisse ha due raggi – un raggio *x* orizzontale e un raggio *y* verticale – il BASIC utilizza l'unico argomento *raggio* in un'istruzione **CIRCLE** come segue: se *aspetto* è minore di uno, allora *raggio* è il raggio delle *x*; se *aspetto* è maggiore o uguale a uno, allora *raggio* è il raggio delle *y*.

Esempio

L'esempio seguente e il suo output spiegano come diversi valori di *aspetto* interessino l'istruzione **CIRCLE**, sia che essa utilizzi l'argomento *raggio* come il raggio delle *x* o come il raggio delle *y* di un'ellisse:

```
SCREEN 1

' Disegna l'ellisse a sinistra:
CIRCLE (60, 100), 80, , , , 3

' Disegna l'ellisse a destra:
CIRCLE (180, 100), 80, , , , 3/10
```

Output

a) *Raggio*
b) *Raggio*
c) *Il rapporto delle dimensioni nell'istruzione CIRCLE è maggiore di 1.*
d) *Il rapporto delle dimensioni nell'istruzione CIRCLE è minore di 1.*

5.4.3 Disegno di archi

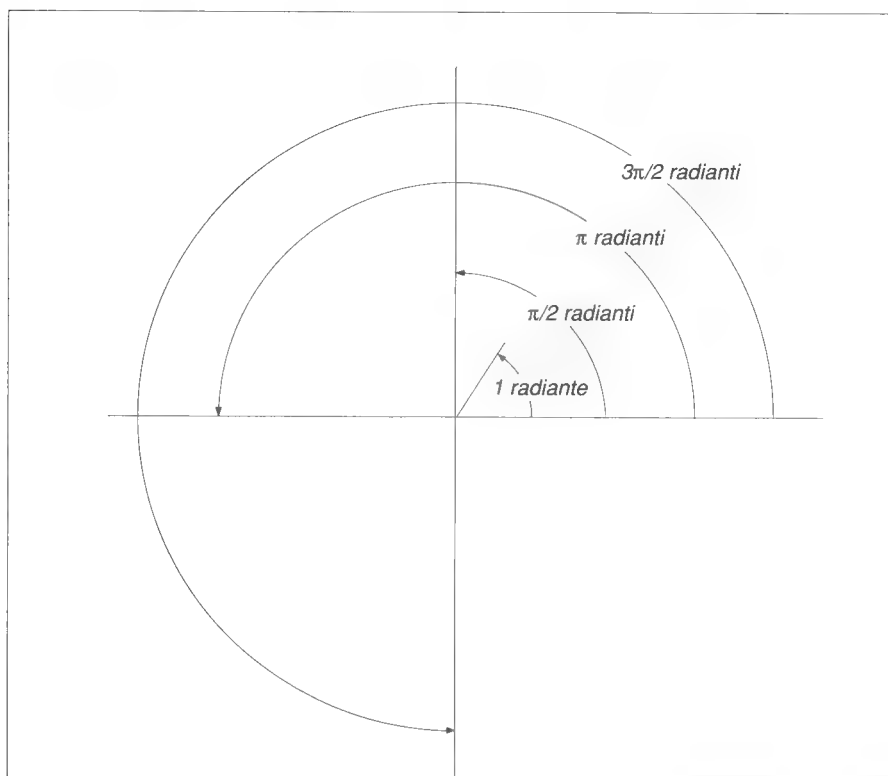
Un arco è un segmento di ellisse, ovvero una breve linea curva. Per comprendere come l'istruzione **CIRCLE** disegna gli archi, bisogna conoscere il modo in cui il BASIC misura gli angoli.

Il BASIC utilizza il radiante come unità di misura per gli angoli, non solo nell'istruzione **CIRCLE**, ma anche nelle funzioni trigonometriche quali **COS, SIN** o **TAN**. (**DRAW** è l'unica istruzione che non utilizza il radiante, in quanto richiede le misure degli angoli in gradi. Per ulteriori informazioni relative a **DRAW**, si consulti la sezione 5.9.)

Il radiante è strettamente collegato al raggio di un cerchio. Infatti la parola "radiante" deriva dalla parola "raggio". La circonferenza di un cerchio è uguale a 2 * π * *raggio*, dove π è uguale approssimativamente a 3,14159265. Allo stesso modo, il numero di radianti in un angolo giro (di 360 gradi) è uguale a 2 * π, o a poco più di 6,28. Per quanto riguarda la misura degli angoli in gradi, ecco alcune comuni equivalenze:

Angoli in gradi	***Angoli in radianti***	
360	2π	(circa 6,283)
180	π	(circa 3,142)
90	π/2	(circa 1,571)
60	π/3	(circa 1,047)

Se si rappresenta sullo schermo il quadrante di un orologio, **CIRCLE** misura gli angoli cominciando dalla posizione "ore 3.00" e ruotando in senso antiorario, come mostra l'illustrazione 5.2.

Illustrazione 5.2 Come si misurano gli angoli per CIRCLE

Un metodo generale per la conversione dei gradi in radianti è di moltiplicare il numero dei gradi per $\pi/180$.

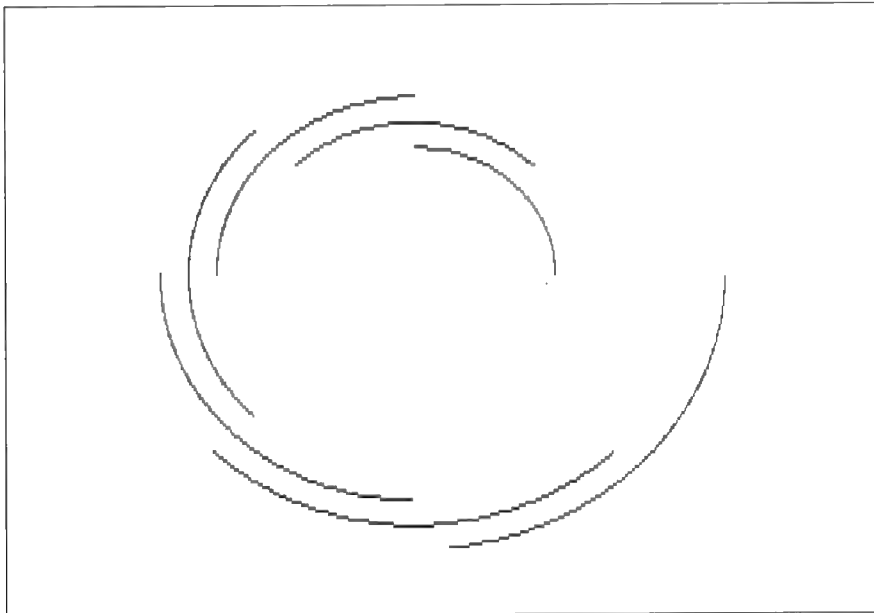
Per disegnare un arco, è sufficiente specificare come argomenti gli angoli che definiscono i limiti dell'arco stesso:

CIRCLE [STEP] (*x*, *y*), *raggio*, [*colore*], *inizio*, *fine* [, *aspetto*]

Esempio

Le istruzioni **CIRCLE** dell'esempio seguente disegnano sette archi, dei quali quello più interno comincia nella posizione "ore 3.00" (0 radianti) e quello più esterno nella posizione "ore 6.00" ($3\pi/2$ radianti), come si evince dall'output:

```
SCREEN 2
CLS

CONST PIGRECO = 3.141592653589        ' Doppia precisione

AngoloIniz = 0
FOR Raggio% = 100 TO 220 STEP 20
   AngoloFin = AngoloIniz + (PIGRECO / 2.01)
   CIRCLE (320, 100), Raggio%, , AngoloIniz, AngoloFin
   AngoloIniz = AngoloIniz + (PIGRECO / 4)
NEXT Raggio%
```

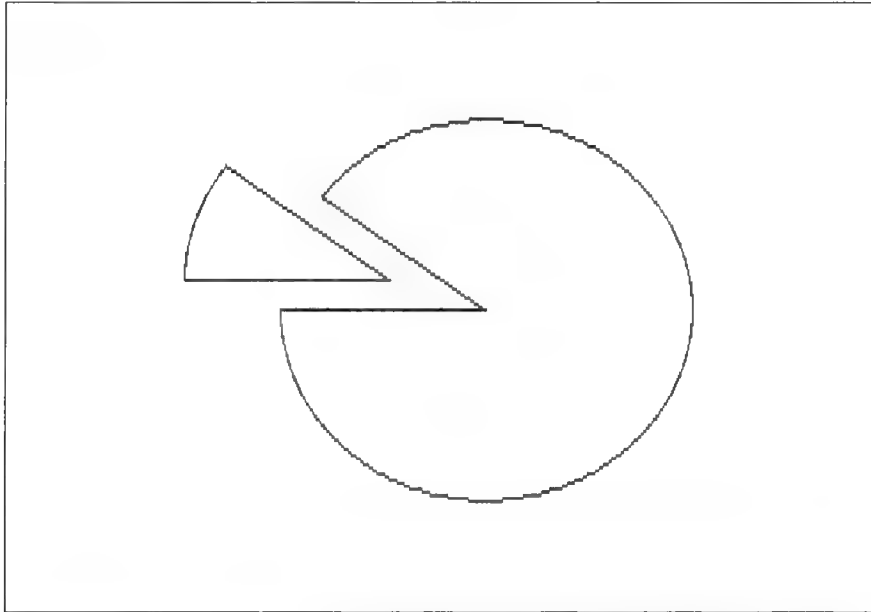
Output

5.4.4 Disegno di grafici a torta

Rendendo negativi gli argomenti *inizio* o *fine* dell'istruzione **CIRCLE**, si può unire il punto di inizio o quello finale dell'arco con il centro del cerchio. Rendendo negativi entrambi gli argomenti, si possono disegnare forme che variano da uno spicchio simile ad una fetta di torta, alla torta stessa con uno spicchio mancante.

Esempio

Questo esempio disegna una forma di torta con uno spicchio tolto:

```
SCREEN 2

CONST RAGGIO = 150, PIGRECO = 3.141592653589#

AngoloIniz = 2.5
AngoloFin = PIGRECO

' Disegna lo spicchio:
CIRCLE (320, 100), RAGGIO, , -AngoloIniz, -AngoloFin

' Scambia i valori degli angoli iniziale e finale:
SWAP AngoloIniz, AngoloFin

' Sposta il centro 10 pixel in giù e 70 pixel a destra,
' poi disegna la torta con la fetta mancante:
CIRCLE STEP(70, 10), RAGGIO, , -AngoloIniz, -AngoloFin
```

Output

5.4.5 Disegno di forme proporzionate alle dimensioni

Come è stato detto nella sezione 5.4.2, "Disegno di ellissi", l'istruzione **CIRCLE** del BASIC regola automaticamente il rapporto delle dimensioni, che determina il modo in cui le figure vengono dimensionate sullo schermo. Con altre istruzioni grafiche, però, è necessario dimensionare esplicitamente le dimensioni orizzontali e verticali per assegnare alle forme le volute proporzioni. Per esempio, sebbene l'istruzione seguente disegni un rettangolo che misura 100 pixel per lato, esso non rassomiglia ad un quadrato:

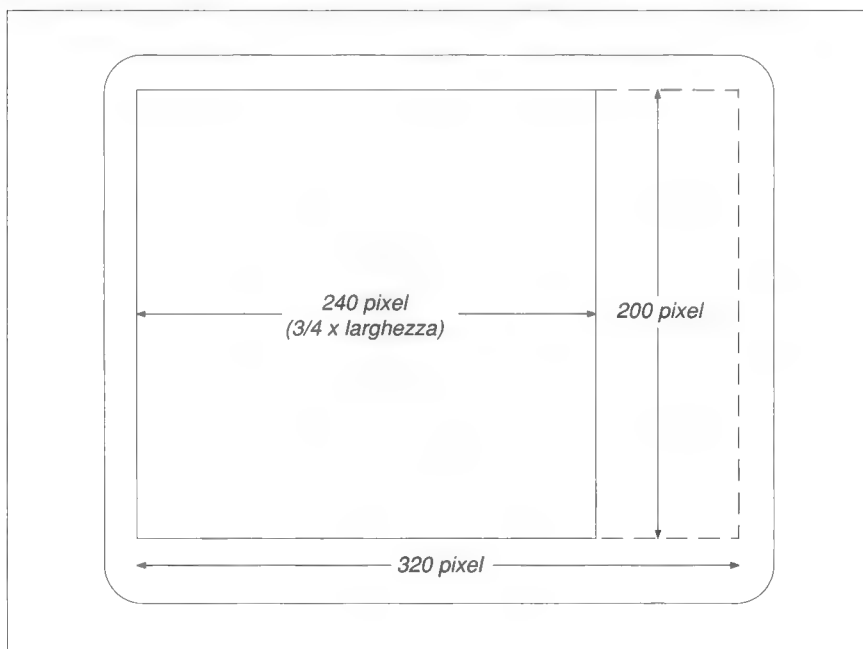
```
SCREEN 1
LINE (0, 0)-(100, 100), , B
```

Non si tratta di un illusione ottica; l'altezza del rettangolo è davvero maggiore della sua larghezza. Ciò avviene perché in modalità schermo 1 c'è uno spazio maggiore tra i pixel verticali che tra quelli orizzontali. Per disegnare un quadrato perfetto, bisogna modificare il rapporto delle dimensioni.

Il rapporto delle dimensioni viene definito nel modo seguente: in una data modalità schermo considerare due linee, una verticale e una orizzontale, della stessa lunghezza visiva. Il rapporto delle dimensioni è dato dal numero di pixel nella linea verticale diviso il numero di pixel in quella orizzontale. Questo rapporto dipende da due fattori:

- Nella maggioranza degli schermi, risulta che una riga orizzontale contiene più pixel di una colonna verticale della stessa lunghezza in tutte le modalità schermo tranne la 11 e la 12.
- Lo schermo standard del monitor di un personal computer ha una larghezza maggiore dell'altezza. Generalmente il rapporto tra larghezza ed altezza è di 4:3.

Per vedere come questi due fattori interagiscono per generare il rapporto delle dimensioni, prendiamo in esame uno schermo dopo un'istruzione **SCREEN 1**, che dà una risoluzione di 320 x 200 pixel. Se si disegna un rettangolo che si estende dal bordo superiore dello schermo fino a quello inferiore, e dal bordo sinistro dello schermo per tre quarti della sua larghezza, si otterrà un quadrato, come nell'illustrazione 5.3

Illustrazione 5.3 Il rapporto delle dimensioni in modalità schermo 1

Come si può vedere dal diagramma, questo quadrato è alto 200 pixel e largo 240 pixel. Il rapporto tra l'altezza del quadrato e la sua larghezza (200/240 o, semplificando, 5/6) è il rapporto delle dimensioni per questa risoluzione di schermo. In altri termini, per disegnare un quadrato in risoluzione 320 x 200, bisogna che la sua altezza in pixel corrisponda ai 5/6 della sua larghezza in pixel, come è illustrato nell'esempio seguente:

```
SCREEN 1             ' Risoluzione di 320 x 200 pixel

' Le dimensioni di questo riquadro sono di 100 pixel
' per 120, per cui il rapporto tra altezza e larghezza
' è 100/120 = 5/6; ne risulterà un quadrato:
LINE (50, 50) -STEP(120, 100), , B
```

La formula per calcolare il rapporto delle dimensioni per una determinata modalità schermo è

(4/3) * (*ypixel*/*xpixel*)

in cui *xpixel* per *ypixel* dà la risoluzione corrente dello schermo. In modalità schermo 1, questa formula rende un rapporto delle dimensioni uguale a (4/3) * (200/320), ovvero 5/6; in modalità schermo 2, il rapporto delle dimensioni è (4/3) * (200/640), ovvero 5/12.

Se si dispone di un monitor in cui il rapporto tra larghezza ed altezza sia diverso da 4:3, utilizzare la seguente formula più generale per il calcolo del rapporto delle dimensioni:

(*larghezzaschermo*/*altezzaschermo*) * (*ypixel*/*xpixel*)

Per disegnare un'ellisse si può utilizzare l'istruzione **CIRCLE** cambiando il valore dell'argomento *aspetto*, come illustrato nella precedente sezione 5.4.2.

5.5 Definizione di una finestra grafica

In tutti gli esempi grafici fino ad ora presentati è stato utilizzato l'intero schermo del monitor come tavolo da disegno, misurando le coordinate assolute dall'angolo superiore sinistro dello schermo.

Tuttavia, utilizzando l'istruzione **VIEW** si può definire una specie di schermo in scala ridotta (noto come "finestra grafica") all'interno dello schermo fisico. Una volta definita la finestra, tutte le successive operazioni grafiche avranno luogo al suo interno. Qualunque output grafico al di fuori dei confini della finestra verrà "tagliato"; cioè, qualsiasi punto le cui coordinate sono al di fuori della finestra non verrà tracciato.

Ciò comporta due vantaggi:

- Con una finestra è semplice modificare la dimensione e la posizione dell'area dello schermo in cui appaiono i grafici.
- Utilizzando **CLS 1**, si può cancellare il contenuto di una finestra, lasciando inalterato il resto dello schermo.

Nota Per informazioni su come creare una "finestra di testo" per l'output stampato sullo schermo, consultare la sezione 3.1.6.

La sintassi generale per **VIEW** (si noti che con **VIEW** non è permessa l'opzione **STEP**) è

VIEW [[SCREEN] (*x1*, *y1*)–(*x2*, *y2*) **[**, **[***colore***]**, **[***bordo***]]]**

dove (*x1*, *y1*) e (*x2*, *y2*) definiscono gli angoli della finestra, utilizzando la sintassi standard del BASIC per i rettangoli (si consulti la sezione 5.3.2.2, "Disegno di riquadri").

Gli argomenti opzionali *colore* e *bordo* permettono di scegliere un colore rispettivamente per l'interno e per i bordi del rettangolo della finestra. La sezione successiva fornisce ulteriori informazioni relative all'impostazione e alla modifica dei colori.

Utilizzando l'istruzione **VIEW** senza argomenti l'intero schermo diventa la finestra. Senza l'opzione **SCREEN**, con l'istruzione **VIEW** tutte le coordinate dei pixel sono relative alla finestra, piuttosto che all'intero schermo. In altri termini, dopo l'istruzione

```
VIEW (50, 60)-(150, 175)
```

un pixel tracciato con

```
PSET (10, 10)
```

sarà visibile, perché si troverà 10 pixel sotto e 10 pixel a destra dell'angolo superiore sinistro della finestra. Notare che le coordinate assolute del pixel (relative all'intero schermo) saranno (50 + 10, 60 + 10), cioè (60, 70).

Al contrario, con l'istruzione **VIEW** e l'opzione **SCREEN** le coordinate rimangono assolute; cioè vengono misurate dai bordi dello schermo, non della finestra. Dopo l'istruzione

```
VIEW SCREEN (50, 60)-(150, 175)
```

un pixel tracciato con

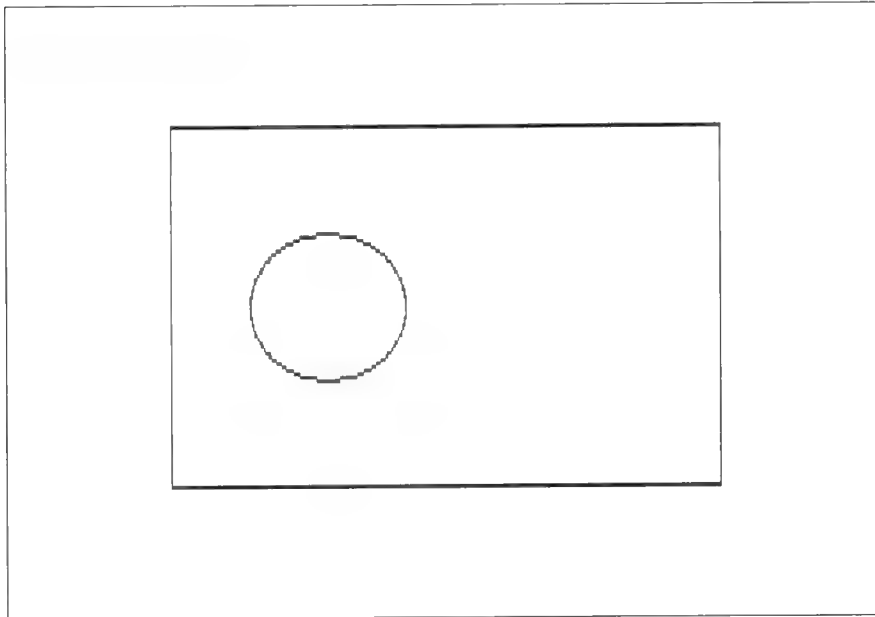
```
PSET (10, 10)
```

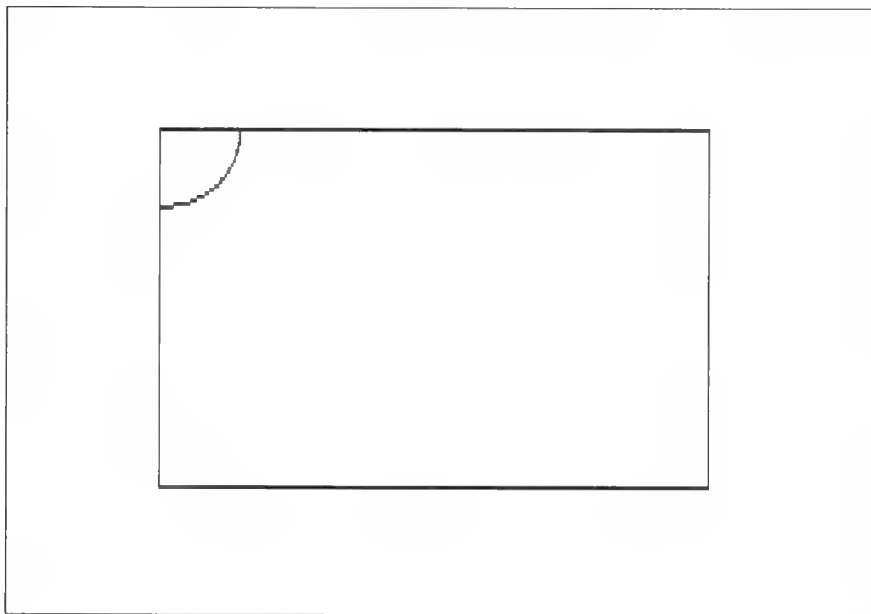
non sarà visibile, perché si troverà 10 pixel sotto e 10 pixel a destra dell'angolo superiore sinistro dello schermo – al di fuori della finestra.

Esempi

L'output dei due esempi seguenti chiarirà ulteriormente la differenza tra **VIEW** e **VIEW SCREEN**:

```
SCREEN 2

VIEW (100, 50)-(450, 150), , 1

' Il centro di questo cerchio ha coordinate assolute
' (100 + 100, 50 + 50), cioè (200, 100):
CIRCLE (100, 50), 50
```

Output utilizzando VIEW

```
SCREEN 2

' Il centro di questo cerchio ha coordinate assolute
' (100, 50), perciò ne appare solo una parte:
VIEW SCREEN (100, 50)-(450, 150), , 1
CIRCLE (100, 50), 50
```

Output utilizzando VIEW SCREEN

Si noti che l'output grafico esterno alla finestra viene tagliato dal bordo della finestra e non appare sullo schermo.

5.6 Ridefinizione delle coordinate di una finestra con WINDOW

Questa sezione spiega come utilizzare l'istruzione **WINDOW** per ridefinire le coordinate dei pixel secondo un sistema di coordinate a scelta.

Nelle sezioni da 5.2 a 5.5, le coordinate utilizzate per individuare i pixel sullo schermo rappresentano distanze fisiche effettive (in pixel) dall'angolo superiore sinistro dello schermo (oppure da quello superiore sinistro della finestra attiva, se è stata definita con un'istruzione **VIEW**). Esse sono note come "coordinate fisiche ". L'origine, o punto di riferimento, per le coordinate fisiche è sempre l'angolo superiore sinistro dello schermo o della finestra, che ha le coordinate (0, 0).

Spostandosi verso il basso e verso destra, i valori delle *x* (coordinate orizzontali) e i valori delle *y* (coordinate verticali) aumentano, come mostra il diagramma in alto nell'illustrazione 5.4. Sebbene questo sia lo schema standard per i monitor, riuscirà insolito a chi è abituato a disegnare grafici con altri sistemi di coordinate. Per esempio, nella griglia cartesiana utilizzata in matematica, i valori delle y aumentano verso l'alto e diminuiscono verso il basso.

Con l'istruzione **WINDOW** del BASIC si può utilizzare qualsiasi sistema di coordinate, e non solo quelle fisiche.

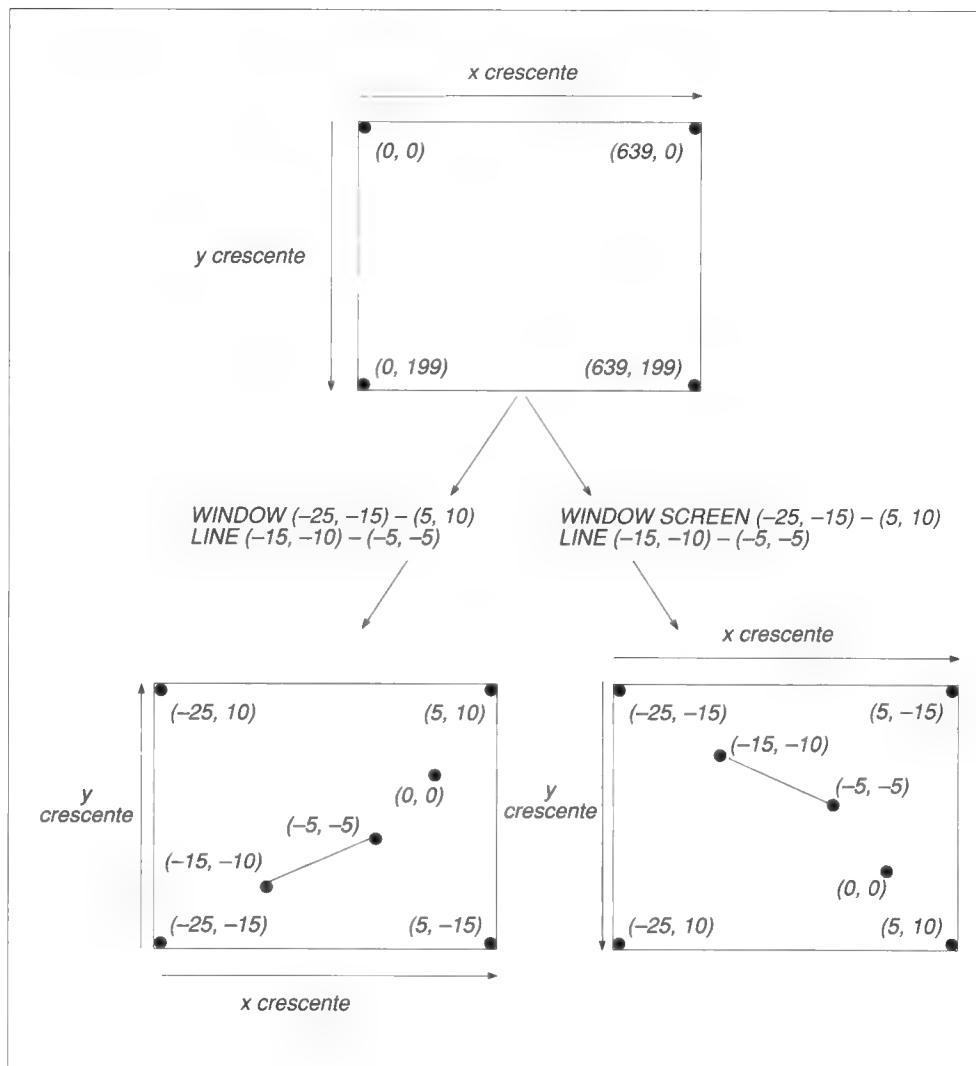
La sintassi generale per **WINDOW** è

WINDOW [[SCREEN] (*x1*, *y1*)-(*x2*, *y2*)**]**

dove *y1*, *y2*, *x1* e *x2* sono numeri reali che determinano rispettivamente il lato superiore, quello inferiore, a sinistra e a destra della finestra. Questi numeri sono noti come "coordinate logiche". Per esempio, l'istruzione seguente reimposta lo schermo in modo che esso sia limitato in alto e in basso dalle righe `y = 10` e `y = -15`, e a sinistra e a destra dalle righe `x = -25` e `x = 5`:

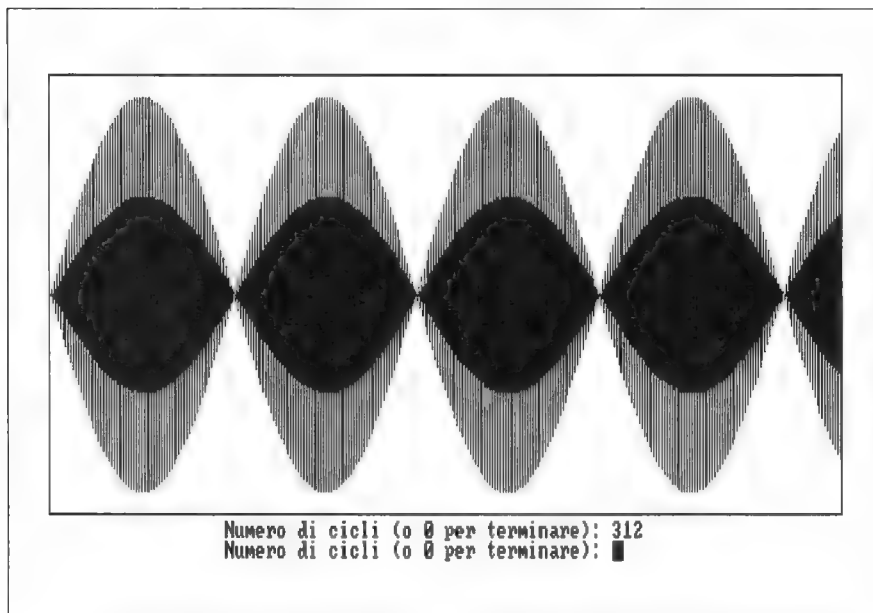
```
WINDOW (-25, -15)-(5, 10)
```

Dopo un'istruzione **WINDOW**, i valori delle y aumentano verso l'alto dello schermo e dopo un'istruzione **WINDOW SCREEN** verso il basso. Le figure in basso nell'illustrazione 5.4 mostrano gli effetti sia di un'istruzione **WINDOW** che di una **WINDOW SCREEN** su una linea disegnata in modalità schermo 2. Si noti anche come entrambe queste istruzioni modifichino le coordinate degli angoli dello schermo. Un'istruzione **WINDOW** senza argomenti ripristina il sistema standard delle coordinate fisiche.

Illustrazione 5.4 WINDOW in confronto a WINDOW SCREEN

Esempio

L'esempio seguente utilizza sia **VIEW** che **WINDOW** per semplificare la stesura di un programma che traccia il grafico della funzione dell'onda sinusoidale per valori di angoli compresi tra 0 e π radianti (cioè tra 0° e 180°). Questo programma si trova nel file chiamato SENO.BAS sui dischi della confezione QuickBASIC.

```
SCREEN 2

' Dimensiona la finestra per il grafico:
VIEW (20, 2)-(620, 172), , 1
CONST PIGRECO = 3.141592653589#

' Dimensiona le coordinate logiche della finestra per
' tracciare il seno da 0 radianti a 2 * PIGRECO radianti:
WINDOW (0, -1.1)-(2 * PIGRECO, 1.1)
Stile% = &HFF00                  ' Per la linea tratteggiata.
VIEW PRINT 23 TO 24              ' Fa scorrere l'output di
                                 ' testo nelle righe 23 e 24.
DO
   PRINT TAB(20);
   INPUT "Numero di cicli (o 0 per terminare): ", Cicli
   CLS
   LINE (2 * PIGRECO, 0)-(0, 0), , , Stile%    ' Traccia
                                               ' l'ascissa.
   IF Cicli > 0 THEN

      ' Inizia da (0, 0) e traccia il grafico:
      FOR X = 0 TO 2 * PIGRECO STEP .01
         Y = SIN(Cicli * X)      ' Calcola la coordinata y.
         LINE -(X, Y)            ' Traccia una linea dal
                                 ' punto precedente al punto
                                 ' corrente.
      NEXT X
   END IF
LOOP WHILE Cicli > 0
```

Output

5.6.1 L'ordine delle coppie di coordinate

L'ordine delle coppie di coordinate in un'istruzione **WINDOW** non è importante, come nelle altre istruzioni grafiche BASIC che definiscono zone rettangolari (**GET**, **LINE**, e **VIEW**). La prima coppia delle seguenti istruzioni produce lo stesso effetto della seconda.

```
VIEW (100, 20)-(300, 120)
WINDOW (-4, -3)-(0, 0)

VIEW (300, 120)-(100, 20)
WINDOW (0, 0)-(-4, -3)
```

5.6.2 Controllo delle coordinate logiche e fisiche

Le funzioni **PMAP** e **POINT** servono per controllare le coordinate fisiche e logiche. La funzione **POINT** (*numero*) indica il posizionamento corrente del cursore grafico, restituendo le coordinate fisiche o quelle logiche (a seconda del valore del *numero*) dell'ultimo punto riportato in un'istruzione grafica. La funzione **PMAP** permette di trasformare le coordinate fisiche in coordinate logiche e viceversa. I valori delle coordinate fisiche restituite da **PMAP** sono sempre relative alla finestra attiva.

Esempi

L'esempio seguente indica i diversi valori restituiti da **POINT** (*numero*) per valori *numero* di 0, 1, 2 o 3:

```
SCREEN 2

' Definisce la finestra per le coordinate logiche:
WINDOW (-10, -30)-(-5, -10)

' Traccia un segmento dal punto con coordinate logiche
' (-9, -28) al punto con coordinate logiche (-6, -24):
LINE (-9, -28)-(-6, -24)

PRINT "Coordinata fisica x dell'ultimo punto = " POINT(0)
PRINT "Coordinata fisica y dell'ultimo punto = " POINT(1)
PRINT
PRINT "Coordinata logica x dell'ultimo punto = " POINT(2)
PRINT "Coordinata logica y dell'ultimo punto = " POINT(3)
END
```

Output

```
Coordinata fisica x dell'ultimo punto = 511
Coordinata fisica y dell'ultimo punto = 139

Coordinata logica x dell'ultimo punto = -6
Coordinata logica y dell'ultimo punto = -24
```

Data l'istruzione **WINDOW** dell'esempio precedente, le quattro seguenti istruzioni **PMAP** visualizzerebbero l'output seguente.

```
' Converte la coordinata x logica -6 in fisica e la stampa:
FisX% = PMAP(-6, 0)
PRINT FisX%

' Converte la coordinata y logica -24 in fisica e la stampa:
FisY% = PMAP(-24, 1)
PRINT FisY%

' Riconverte la fisica x in logica x e la stampa:
LogX% = PMAP(FisX%, 2)
PRINT LogX%

' Riconverte la fisica y in logica y e la stampa:
LogY% = PMAP(FisY%, 3)
PRINT LogY%
```

Output

```
 511
 139
-6
-24
```

5.7 Utilizzo dei colori

Se si dispone di scheda CGA, si può scegliere soltanto tra queste due modalità grafiche:

- La modalità schermo 2, alta risoluzione (640 x 200 pixel), un solo colore di primo piano e uno di sfondo. Essa è detta "monocromatica" perché ogni output grafico presenta lo stesso colore.
- La modalità schermo 1, media risoluzione (320 x 200 pixel), quattro colori di primo piano e sedici di sfondo.

Esiste pertanto un compromesso tra gamma colori e nitidezza in queste due modalità schermo supportate dalla maggior parte delle schede grafiche a colori. Con un sistema con maggiori capacità grafiche che non la scheda CGA, non sarà necessario sacrificare la nitidezza per avere una piena gamma di colori. Tuttavia, questa sezione si indirizza soprattutto alle modalità schermo 1 e 2.

5.7.1 Scelta di un colore per output grafico

L'elenco seguente spiega dove inserire l'argomento *colore* nelle istruzioni grafiche trattate nelle precedenti sezioni di questo capitolo. Questo elenco presenta anche altre opzioni (quali **BF** con l'istruzione **LINE** o *bordo* con l'istruzione **VIEW**) che possono avere diversi colori. (Si noti che qui non si fornisce la sintassi completa per alcune di queste istruzioni. Il presente sommario ha solo lo scopo di dimostrare come utilizzare l'opzione *colore* in quelle istruzioni che l'accettano.)

PSET (*x*, *y*), *colore*
PRESET (*x*, *y*), *colore*
LINE (*x1*, *y1*)–(*x2*, *y2*), *colore* [, **B** [**F**]]
CIRCLE (*x*, *y*), *raggio*, *colore*
VIEW (*x1*, *y1*)–(*x2*, *y2*), *colore*, *bordo*

In modalità schermo 1, l'argomento *colore* è un'espressione numerica con valore 0, 1, 2, o 3. Ciascuno di questi valori, noto come "attributo", rappresenta un colore diverso, come si evince nel seguente programma:

```
' Traccia una linea "invisibile" (del colore di sfondo):
LINE (10, 10)-(310, 10), 0

' Traccia una linea azzurra:
LINE (10, 30)-(310, 30), 1

' Traccia una linea magenta:
LINE (10, 50)-(310, 50), 2

' Traccia una linea bianca:
LINE (10, 70)-(310, 70), 3
END
```

Come è stato osservato nei commenti dell'esempio precedente, un valore di 0 per *colore* genera un'output invisibile perché sempre uguale al colore di sfondo. A prima vista, questo non sembra un valore utile, ma invece può servire per cancellare una figura senza cancellare l'intero schermo o finestra, come nell'esempio seguente:

```
SCREEN 1
CIRCLE (100, 100), 80, 2, , , 3      ' Traccia un'ellisse.
Pausa$ = INPUT$(1)                   ' Attende una pressione
                                     ' di tasto.
CIRCLE (100, 100), 80, 0, , , 3      ' Cancella l'ellisse.
```

5.7.2 Sostituzione del colore di primo piano o del colore di sfondo

Come si è visto, per l'output grafico si possono utilizzare quattro diversi colori di primo piano. I colori di sfondo disponibili in modalità schermo 1 sono invece 16 in tutto.

Inoltre, si può modificare il colore di primo piano utilizzando una diversa "palette". In modalità schermo 1, ci sono due palette, ovvero gruppi di quattro colori. Ciascuna palette assegna un colore diverso allo stesso attributo; così, per esempio, nella palette 1 (quella predefinita) il colore associato all'attributo 2 è il magenta, mentre nella palette 0 il colore associato all'attributo 2 è il rosso. Se si dispone di una scheda CGA, questi colori sono predeterminati per ciascuna palette; cioè, il colore assegnato al numero 2 nella palette 1 è sempre il magenta, mentre quello assegnato al numero 2 nella palette 0 è sempre il rosso.

Qualora si disponga di una scheda EGA o VGA, utilizzando l'istruzione **PALETTE** si può scegliere il colore determinato da qualunque attributo. Per mezzo di istruzioni **PALETTE** con argomenti diversi, è possibile, per esempio, rendere il colore determinato dall'attributo 1 una volta verde e un'altra marrone. (Per ulteriori informazioni circa il modo in cui riassegnare i colori, consultare la sezione 5.7.3, "Sostituzione dei colori con PALETTE e PALETTE USING".)

In modalità schermo 1, l'istruzione **COLOR** permette di controllare sia il colore di sfondo che la palette per i colori di primo piano. La sintassi per **COLOR** in modalità schermo 1 è la seguente:

COLOR [*sfondo*] [, *palette*]

L'argomento *sfondo* è un'espressione numerica da 0 a 15, e *palette* è un'espressione numerica equivalente a 0 oppure a 1.

La tabella 5.1 indica i 4 colori di primo piano in ciascuna delle due palette, mentre la tabella 5.2 contiene i 16 colori di sfondo.

Tabella 5.1 Palette dei colori in modalità schermo 1

Numero del colore di primo piano	***Colore nella palette 0***	***Colore nella palette 1***
0	Colore di sfondo corrente	Colore di sfondo corrente
1	Verde	Azzurro (verde bluastro)
2	Rosso	Magenta (viola chiaro)
3	Marrone	Bianco (su alcuni monitor, grigio chiaro)

Tabella 5.2 Colori di sfondo in modalità schermo 1

Numero del colore di sfondo	*Colore*
0	Nero
1	Blu
2	Verde
3	Azzurro
4	Rosso
5	Magenta
6	Marrone (su alcuni monitor, giallo scuro)
7	Bianco (su alcuni monitor, grigio chiaro)
8	Grigio scuro (su alcuni monitor, nero)
9	Blu chiaro
10	Verde chiaro
11	Azzurro chiaro
12	Rosso chiaro
13	Magenta chiaro
14	Giallo chiaro (su alcuni monitor può presentare sfumature verdastre)
15	Bianco intenso oppure grigio chiarissimo

Esempio

Il programma seguente illustra tutte le combinazioni delle due palette di colori con i 16 diversi colori di sfondo dello schermo. Esso si trova nel file COLORI.BAS sui dischi della confezione QuickBASIC.

```
SCREEN 1

Esc$ = CHR$(27)

' Disegna tre quadrati e ne dipinge l'interno con
' tre colori diversi:
FOR ColorVal = 1 TO 3
   LINE (X, Y)-STEP(60, 50), ColorVal, BF
```

```
   X = X + 61
   Y = Y + 51
NEXT ColorVal

LOCATE 21, 1
PRINT "Premere <ESC> per uscire."
PRINT "Premere un tasto per continuare."

' Limita ogni ulteriore visualizzazione di testo alla
' riga 23:
VIEW PRINT 23 TO 23

DO
   PaletteVal = 1
   DO

      ' PaletteVal è o uno o zero:
      PaletteVal = 1 - PaletteVal

      ' Imposta il colore dello sfondo e sceglie la palette:
      COLOR SfondoVal, PaletteVal
      PRINT "Sfondo ="; SfondoVal; "Palette ="; PaletteVal;

      Pausa$ = INPUT$(1)          ' Aspetta la pressione di un
                                  ' tasto.
      PRINT

   ' Esce dal ciclo se entrambe le palette sono state
   ' visualizzate, oppure se l'utente ha premuto il tasto
   ' ESC:
   LOOP UNTIL PaletteVal = 1 OR Pausa$ = Esc$

   SfondoVal = SfondoVal + 1

' Esce dal ciclo se tutti e sedici i colori di sfondo sono
' stati visualizzati, oppure se l'utente ha premuto il
' tasto ESC:
LOOP UNTIL SfondoVal > 15 OR Pausa$ = Esc$

SCREEN 0                          ' Ripristina la modalità
WIDTH 80                          ' testo e la larghezza
                                  ' schermo di 80 colonne.
```

5.7.3 Sostituzione dei colori con PALETTE e PALETTE USING

Nella precedente sezione è stato illustrato come modificare il colore determinato da un attributo, semplicemente specificando una diversa palette nell'istruzione **COLOR**. Si è però limitati a due palette con colori fissi, in ciascuna delle quali vi sono solo 4 colori. Inoltre, ciascun attributo può determinare soltanto due colori possibili; per esempio, l'attributo 1 può significare solo verde o azzurro.

Una scheda EGA o VGA offre maggiori alternative. (Qualora non si disponga di una scheda EGA o VGA, si può saltare questa sezione.) Per esempio, a seconda della quantità di memoria video disponibile, con una scheda VGA è possibile scegliere da una palette fino a 256K colori (256000) ed assegnarli a 256 attributi diversi. Anche una scheda EGA permette di visualizzare 16 colori diversi da una palette di 64 colori.

Al contrario dell'istruzione **COLOR**, le istruzioni **PALETTE** e **PALETTE USING** permettono di manipolare con maggiore flessibilità la palette dei colori disponibili. Utilizzando queste istruzioni, si può assegnare qualunque colore della palette a qualsiasi attributo. Per esempio, dopo l'istruzione seguente, l'output delle istruzioni grafiche con attributo 4 appare in magenta chiaro (colore 13):

```
PALETTE 4, 13
```

Il cambiamento del colore è istantaneo ed interessa non solo le istruzioni grafiche successive, ma anche qualunque output già presente sullo schermo. In altri termini, è possibile disegnare e colorare lo schermo, quindi cambiare palette per cambiare il colore all'istante, come illustra l'esempio che segue:

```
SCREEN 8

LINE (50, 50)-(150, 150), 4    ' Traccia una linea in rosso.
SLEEP 1                        ' Pausa momentanea.
PALETTE 4, 13                  ' L'attributo 4 ora è
                               ' assegnato al colore 13,
                               ' la linea tracciata prima
                               ' ora è magenta chiaro.
```

Mediante l'opzione **USING** dell'istruzione **PALETTE**, si possono cambiare contemporaneamente tutti i colori assegnati ad ogni attributo.

Esempio

In questo esempio l'istruzione **PALETTE USING** crea l'illusione del movimento sullo schermo, mediante la rotazione continua dei colori visualizzati dagli attributi da 1 a 15. Questo programma si trova nel file PALETTE.BAS sui dischi della confezione QuickBASIC.

```
DECLARE SUB InizPalette()
DECLARE SUB CambiaPalette()
DECLARE SUB DisegnaEllissi()
DEFINT A-Z

DIM SHARED MatrPalette(15)

SCREEN 8                    ' Risoluzione 640 x 200; 16 colori

InizPalette
DisegnaEllissi

DO
   CambiaPalette
LOOP WHILE INKEY$ = ""      ' Sposta la palette finché non
                            ' viene premuto un tasto.

END

' ==================== CambiaPalette =====================
'  Questa procedura sposta la palette di una unità a
'  ogni chiamata. Per esempio, dopo la prima chiamata a
'  CambiaPalette, MatrPalette(1) = 2, MatrPalette(2) = 3,
'  . . . , MatrPalette(14) = 15, e MatrPalette(15) = 1
' ========================================================

SUB CambiaPalette STATIC
   FOR I = 1 TO 15
      MatrPalette(I) = (MatrPalette(I) MOD 15) + 1
   NEXT I

   ' Sposta il colore visualizzato dagli attributi 1 - 15:
   PALETTE USING MatrPalette(0)
END SUB
```

```
' ==================== DisegnaEllissi =====================
'  Questa procedura disegna quindici ellissi concentriche e
'  ne colora l'interno con colori diversi.
' ==========================================================

SUB DisegnaEllissi STATIC
   CONST ASPECT = 1 / 3
   FOR ColorVal = 15 TO 1 STEP -1
      Radius = 20 * ColorVal
      CIRCLE (320, 100), Radius, ColorVal, , , ASPECT
      PAINT (320, 100), ColorVal
   NEXT
END SUB

' ===================== InizPalette =======================
'  Questa procedura inizializza la matrice usata per
'  cambiare la palette.
' ==========================================================

SUB InizPalette STATIC
   FOR I = 0 TO 15
      MatrPalette(I) = I
   NEXT I
END SUB
```

5.8 Forme a colori

Con il metodo della sezione 5.3.2.2 si può disegnare un riquadro con l'opzione **B** dell'istruzione **LINE** e colorarlo con l'opzione **F**:

```
SCREEN 1
' Disegna un quadrato, poi ne colora l'interno con
' il colore 1 (azzurro nella palette predefinita):
LINE (50, 50)-(110, 100), 1, BF
```

Con l'istruzione **PAINT** del BASIC, si può colorare qualsiasi figura chiusa con qualsiasi colore. **PAINT** permette inoltre di colorare delle figure con motivi svariati, quali strisce o quadratini, come viene illustrato nella sezione 5.8.2, "Motivi a colori".

5.8.1 Disegni a colori

Per colorare tutta di un colore una forma chiusa, si può utilizzare questa forma dell'istruzione **PAINT**:

PAINT [STEP] (*x*, *y*) **[,** [*interno*]**,** [*bordo*]**]**

Qui, *x* e *y* sono le coordinate di un punto all'interno della figura che si vuole colorare; *interno* rappresenta il numero del colore scelto, mentre *bordo* è il numero del colore utilizzato per il contorno della figura.

Ad esempio, le righe del programma seguente disegnano un cerchio azzurro con interno magenta:

```
SCREEN 1
CIRCLE (160, 100), 50, 1
PAINT (160, 100), 2, 1
```

Quando si colora una figura, è necessario rispettare alcune regole:

- Le coordinate fornite nell'istruzione **PAINT** devono riferirsi a un punto all'interno della figura.

  Ad esempio, ciascuna delle seguenti istruzioni avrebbe lo stesso effetto dell'istruzione **PAINT** nell'esempio precedente, perché tutte le coordinate identificano punti all'interno del cerchio:

  ```
  PAINT (150, 90), 2, 1
  PAINT (170, 110), 2, 1
  PAINT (180, 80), 2, 1
  ```

  Al contrario, poiché (5, 5) identifica un punto esterno al cerchio, l'istruzione successiva colorerebbe tutto lo schermo tranne l'interno del cerchio, che conserverebbe il colore corrente di sfondo:

  ```
  PAINT (5, 5), 2, 1
  ```

  Se le coordinate in un'istruzione **PAINT** determinano un punto esattamente sul bordo della figura, allora non avviene alcuna colorazione:

  ```
  LINE (50, 50)-(150, 150), , B      ' Disegna un riquadro.
  PAINT (50, 100)                   ' Il punto (50, 100) si trova sul
                                    ' lato superiore del riquadro;
                                    ' non avviene colorazione.
  ```

- La figura deve essere completamente chiusa; altrimenti il colore si "espanderà" colorando l'intero schermo o la finestra (o qualunque figura più grande che racchiuda la prima).

continua

Ad esempio, nel programma seguente, l'istruzione **CIRCLE** disegna un'ellisse non proprio completa (sul lato destro c'è una piccola apertura), e l'istruzione **LINE** la racchiude in un riquadro. Anche se la colorazione comincia nell'interno dell'ellisse, il colore si espande attraverso l'apertura e colora l'intero riquadro.

```
SCREEN 2

CONST PIGRECO = 3.141592653589#
CIRCLE (300, 100), 80, , 0, 1.9 * PIGRECO, 3
LINE (200, 10)-(400, 190), , B
PAINT (300, 100)
```

- Se si utilizzano due colori diversi per l'interno ed il bordo della figura, è necessaria l'opzione *bordo* per indicare a **PAINT** dove terminare la colorazione.

Ad esempio, il programma che segue disegna il contorno di un triangolo in verde (attributo 1 nella palette 0) e cerca poi di colorarne l'interno in rosso (attributo 2). Tuttavia, poiché l'istruzione **PAINT** non indica la fine della colorazione, anche lo schermo sarà colorato in rosso:

```
SCREEN 1
COLOR , 0

LINE (10, 25)-(310, 25), 1
LINE -(160, 175), 1
LINE -(10, 25), 1

PAINT (160, 100), 2
```

Con questa modifica all'istruzione **PAINT** l'espandersi del rosso (colore 2) si blocca quando incontra il verde (colore 1):

```
PAINT (160, 100), 2, 1
```

Si noti che non è necessario specificare il colore del bordo nell'istruzione **PAINT** se il colore è lo stesso del bordo.

```
LINE (10, 25)-(310, 25), 1
LINE -(160, 175), 1
LINE -(10, 25), 1

PAINT (160, 100), 1
```

5.8.2 Motivi a colori: creazione di motivi

Per colorare con un motivo qualunque figura chiusa, si può utilizzare l'istruzione **PAINT**: questo procedimento è noto come "creazione di motivi". La "casella motivo" è l'unità modulare di base del motivo; questo risulta riempiendo la figura di copie della casella motivo, analogamente alla disposizione di mattonelle su un pavimento. Nella creazione di motivi, l'argomento *interno* nella sintassi di **PAINT** è un'espressione a stringa, piuttosto che un numero. Sebbene *interno* possa essere una qualunque espressione a stringa, per definire le caselle motivo conviene utilizzare la seguente forma per *interno*:

CHR$** (*arg1*) + **CHR$ (*arg2*) + **CHR$** (*arg3*) + ... + **CHR$** (*argn*)

Qui, *arg1*, *arg2*, e così via sono interi a 8 bit. Le sezioni 5.8.2.2–5.8.2.4 spiegano come calcolare questi interi a 8 bit.

5.8.2.1 Dimensioni della casella motivo in diverse modalità schermo

Ciascuna casella motivo è composta da una griglia rettangolare di pixel. Tale griglia può contenere fino a 64 righe in tutte le modalità schermo; tuttavia, il numero di pixel per riga dipende dalla modalità schermo.

La ragione per cui la lunghezza delle righe delle caselle motivo varia a seconda della modalità schermo è la seguente: sebbene il numero di bit in ciascuna riga sia fisso (è 8, la lunghezza di un intero), il numero dei pixel rappresentati dagli 8 bit diminuisce con l'aumentare del numero degli attributi colore disponibili nella modalità schermo. Ad esempio, nella modalità schermo 2, che dispone soltanto di 1 attributo colore, il numero di bit per pixel è 1. Esso è invece 2 per la modalità schermo 1 (4 attributi colore); e nella modalità schermo EGA 7, (16 attributi), il numero di bit per pixel è 4. Il calcolo dei bit per pixel in una data modalità schermo si può eseguire applicando la formula seguente:

bit-per-pixel = $\log_2$ (*numattributi*)

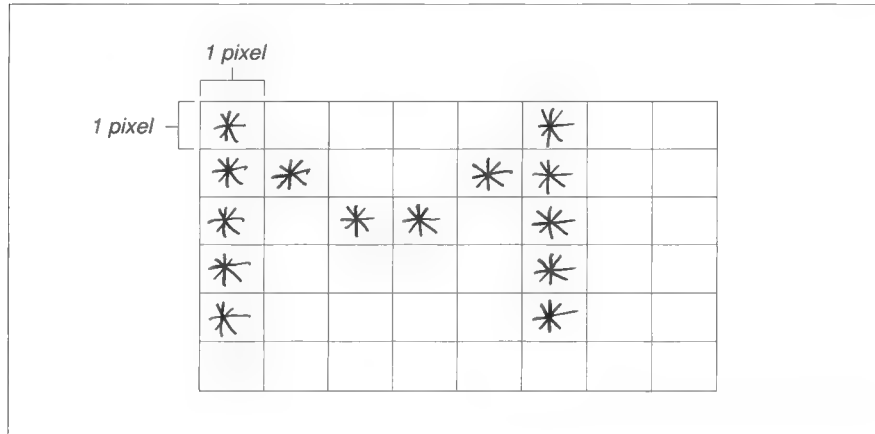
dove *numattributi* rappresenta il numero degli attributi colore nella modalità schermo (tale informazione è contenuta nella guida in linea Consulente QB).

La lunghezza di una riga della casella è dunque di 8 pixel in modalità schermo 2 (cioè 8 bit diviso 1 bit per pixel), e soltanto di 4 pixel in modalità schermo 1 (8 bit diviso 2 bit per pixel).

Nelle tre sezioni seguenti viene illustrato il procedimento della creazione di una casella motivo. In particolare, la sezione 5.8.2.2 spiega come creare un motivo monocromatico in modalità schermo 2. La sezione successiva riguarda la creazione di un motivo policromo in modalità schermo 1; ed infine, per chi disponesse di scheda EGA, la sezione 5.8.2.4 spiega come creare un motivo policromo in modalità schermo 8.

5.8.2.2 Creazione di un motivo monocromatico in modalità schermo 1

I punti seguenti illustrano il modo in cui definire e utilizzare un motivo di caselle simile a una lettera "M":

1. Disegnare su un foglio il motivo per la casella, usando una griglia a 8 colonne e con il numero di righe necessarie (massimo 64). In quest'esempio, la casella è composta da 6 righe e l'asterisco (*) nella casella indica che il pixel è da accendere:

2. Successivamente, convertire ciascuna riga di pixel in un numero a 8 bit, nel quale 1 indica che il pixel è acceso e 0 che è spento:

3. Convertire i numeri binari del punto 2 in interi esadecimali:

```
10000100 = &H84
11001100 = &HCC
10110100 = &HB4
10000100 = &H84
10000100 = &H84
00000000 = &H00
```

Questi interi non devono necessariamente essere esadecimali; essi possono essere decimali od ottali, ma la conversione da binario a esadecimale risulta la più semplice. Per operare tale conversione, leggere il numero binario da destra verso sinistra: ogni gruppo di quattro cifre verrà convertito nel suo equivalente esadecimale, come si vede qui:

La tabella 5.3 elenca sequenze binarie di quattro bit e i loro equivalenti esadecimali.

4. Creare una stringa concatenando i caratteri con i valori ASCII del punto 3 (per ottenere tali caratteri, utilizzare la funzione **CHR$**):

```
Casella$ = CHR$(&H84) + CHR$(&HCC) + CHR$(&HB4)
Casella$ = Casella$ + CHR$(&H84) + CHR$(&H84) + CHR$(&H00)
```

5. Disegnare una figura e colorarne l'interno, per mezzo dell'istruzione **PAINT** e l'argomento a stringa del punto 4:

```
PAINT (X, Y), Casella$
```

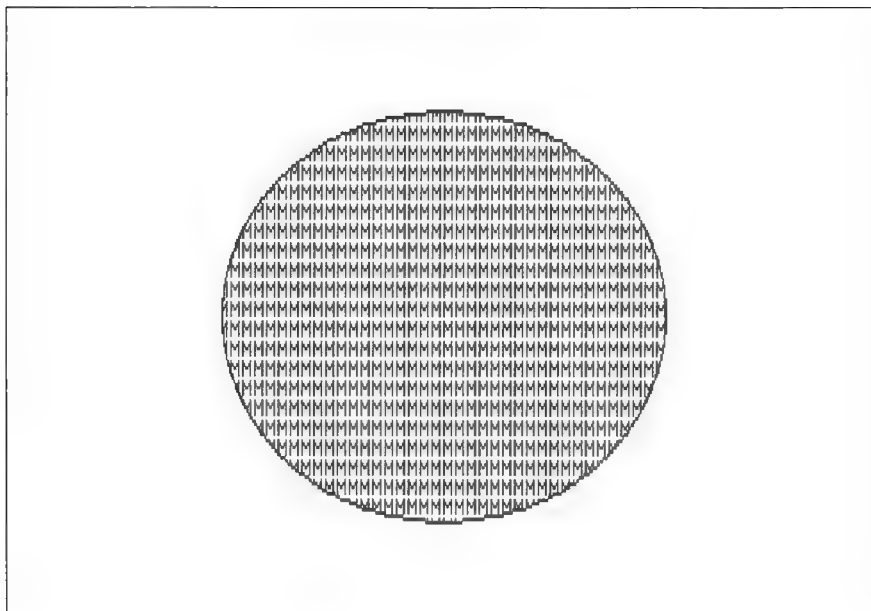
Tabella 5.3 Conversione da binario a esadecimale

Numero binario	*Numero esadecimale*
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

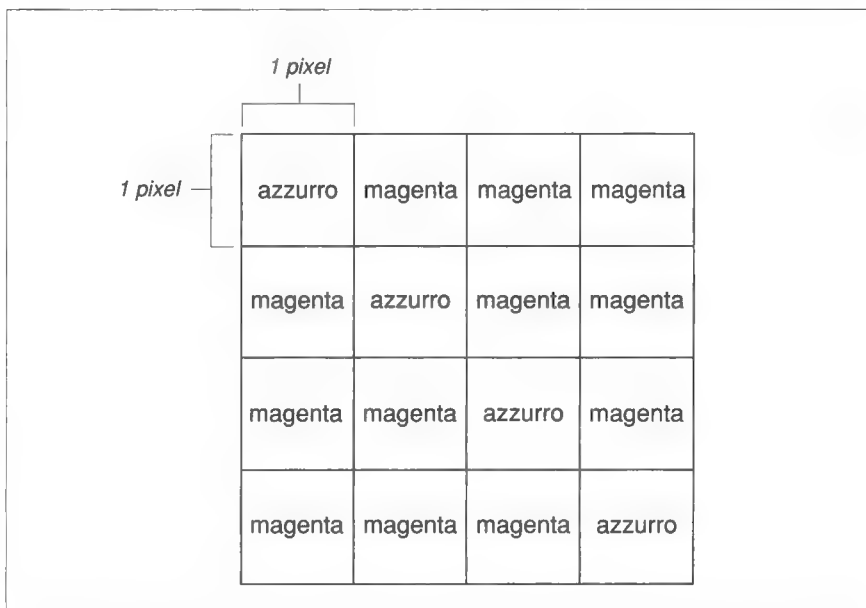
Esempio

L'esempio seguente disegna un cerchio e poi ne colora l'interno con il motivo creato nei punti precedenti:

```
SCREEN 2
CLS
Casella$ = CHR$(&H84) + CHR$(&HCC) + CHR$(&HB4)
Casella$ = Casella$ + CHR$(&H84) + CHR$(&H84) + CHR$(&H00)
CIRCLE STEP (0, 0), 150
PAINT STEP (0, 0), Casella$
```

Output

5.8.2.3 Creazione di un motivo policromo in modalità schermo 1

I punti seguenti indicano come creare un motivo policromo formato da strisce diagonali alternate in azzurro e magenta (o verde e rosso in palette 0):

1. Disegnare su un foglio il motivo per la casella usando una griglia con quattro colonne e il numero di righe necessario, fino a un massimo di 64. (Le colonne sono quattro perché ciascuna riga di pixel viene memorizzata in un intero di 8 bit e ogni pixel in modalità schermo 1 richiede due bit.) Nel diagramma illustrato in questo esempio, la casella è composta da quattro righe:

2. Convertire i colori nei loro rispettivi numeri colore in notazione binaria, come illustrato (assicurarsi di utilizzare valori a due bit: 1 equivale al numero binario 01 e 2 al numero binario 10):

2 bit

01	10	10	10
10	01	10	10
10	10	01	10
10	10	10	01

3. Convertire i numeri binari del punto 2 in interi esadecimali:

```
01101010 = &H6A
10011010 = &H9A
10100110 = &HA6
10101001 = &HA9
```

4. Creare una stringa concatenando i caratteri con i valori ASCII del punto 3 (utilizzare la funzione **CHR$** per ottenere questi caratteri):

```
Casella$ = CHR$(&H6A) + CHR$(&H9A) + CHR$(&HA6) + CHR$(&HA9)
```

5. Disegnare una figura e colorarne l'interno utilizzando **PAINT** e l'argomento a stringa del punto 4:

```
PAINT (X, Y), Casella$
```

Il programma seguente disegna un triangolo e ne colora l'interno con il motivo creato nei punti precedenti:

```
SCREEN 1

' Definisce un motivo:
Casella$ = CHR$(&H6A) + CHR$(&H9A) + CHR$(&HA6) + CHR$(&HA9)

' Traccia un triangolo bianco (colore 3):
LINE (10, 25)-(310, 25)
LINE -(160, 175)
LINE -(10, 25)

' Riempie l'interno del triangolo con il motivo:
PAINT (160, 100), Casella$
```

Se per il contorno della figura si utilizza lo stesso colore del motivo, bisogna specificare l'argomento *bordo* con **PAINT**; altrimenti, il motivo supererà il contorno della figura:

```
SCREEN 1

' Definisce un motivo:
Casella$ = CHR$(&H6A) + CHR$(&H9A) + CHR$(&HA6) + CHR$(&HA9)

' Traccia un triangolo magenta (colore 2):
LINE (10, 25)-(310, 25), 2
LINE -(160, 175), 2
LINE -(10, 25), 2

' Riempie l'interno del triangolo con il motivo,
' con l'argomento bordo (, 2) per indicare a PAINT
' dove fermarsi:
PAINT (160, 100), Casella$, 2
```

Dopo aver colorato una figura con un colore o con un motivo, è possibile ricolorare la stessa figura, o parte di essa, con un nuovo motivo. Se questo però contiene due o più righe adiacenti dello stesso colore dello sfondo della figura, la creazione del motivo non funzionerà. Il motivo comincerà ad espandersi, si vedrà circondato da pixel uguali a due o più delle proprie righe, e si fermerà.

Si può ovviare a questo problema utilizzando l'argomento *sfondo* con **PAINT** se nel motivo ci sono al massimo due righe adiacenti del colore dello sfondo. **PAINT** con *sfondo* ha la seguente sintassi:

PAINT [STEP] (*x*, *y*) [, [*interno*] [, [*contorno*] [, *sfondo*]]]

L'argomento *sfondo* è un carattere a stringa della forma **CHR$** (*n*) che specifica quali righe nella casella motivo hanno il colore dello sfondo.

In sostanza, *sfondo* comanda a **PAINT** di saltare queste righe durante la ricolorazione della figura. L'esempio seguente spiega il modo in cui ciò avviene:

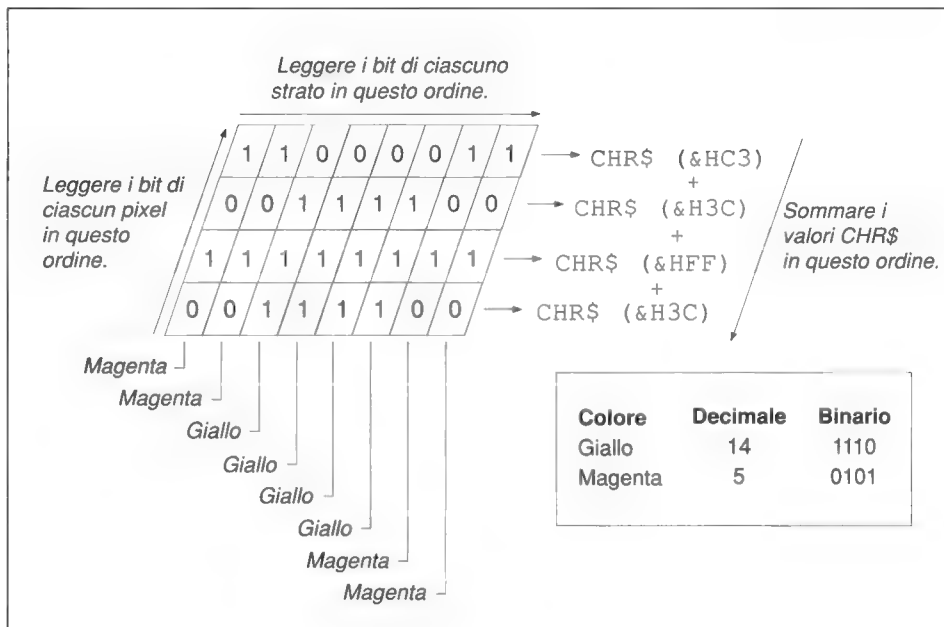
```
SCREEN 1

' Definisce un motivo con 2 linee azzurre, 2 linee
' magenta e 2 linee bianche:
Casella$ = CHR$(&H55) + CHR$(&H55) + CHR$(&HAA)
Casella$ = Casella$ + CHR$(&HAA) + CHR$(&HFF) + CHR$(&HFF)

' Traccia un triangolo bianco (colore 3):
LINE (10, 25)-(310, 25)
LINE -(160, 175)
LINE -(10, 25)

' Ne riempie l'interno in magenta:
PAINT (160, 100), 2, 3

' Attende la pressione di un tasto:
Pausa$ = INPUT$(1)

' Dato che lo sfondo è già magenta, CHR$(&HAA) comanda a
' PAINT di saltare le linee magenta della casella motivo:
PAINT (160, 100), Casella$, , CHR$(&HAA)
```

5.8.2.4 Creazione di un motivo policromo in modalità schermo 8

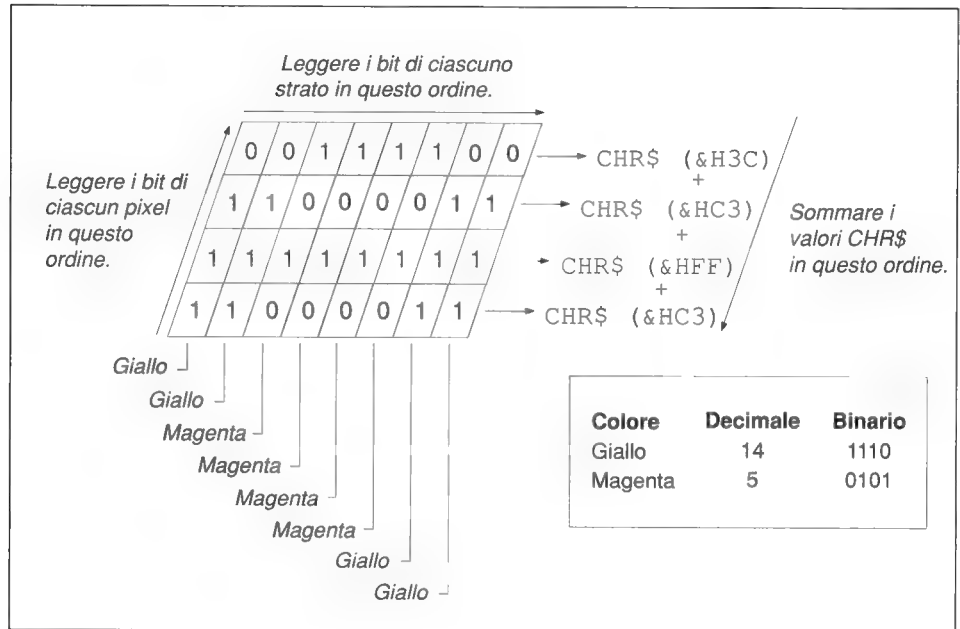
Nelle modalità schermo EGA e VGA, è necessario più di un intero a 8 bit per definire una riga in una casella motivo. In queste modalità schermo, una riga è composta da vari strati di interi a 8 bit. Ciò avviene perché un pixel è rappresentato in modo tridimensionale, con una pila di "strati di bit", piuttosto che sequenzialmente in un unico strato come nelle modalità schermo 1 e 2. Per esempio, in modalità schermo 8 ci sono quattro di questi strati di bit; ciascuno dei quattro bit per pixel si trova in uno strato diverso.

I seguenti punti indicano il processo di creazione di un motivo policromo composto da righe alternate gialle e magenta. Si noti come ciascuna riga nella casella motivo sia rappresentata da quattro byte paralleli.

1. Definire una riga di pixel nella casella motivo. Ciascun pixel della riga prende quattro bit, e ciascun bit si trova in uno strato diverso, come nell'illustrazione:

Colore	Decimale	Binario
Giallo	14	1110
Magenta	5	0101

Sommare i valori **CHR$** di tutti e quattro gli strati di bit, ottenendo un byte della casella. Questa riga viene ripetuta nella casella motivo, per cui:

Riga$ (1) = Riga$ (2) =
CHR$** (&HC3) + **CHR$ (&H3C) + **CHR$** (&HFF) + **CHR$** (&H3C)

2. Definire un'altra riga di pixel nella casella motivo come segue:

Anche questa riga viene ripetuta nella casella motivo, per cui:

Riga$ (3) = Riga$ (4) =
CHR$** (&H3C) + **CHR$ (&HC3) + **CHR$** (&HFF) + **CHR$** (&HC3)

Esempio

L'esempio seguente disegna una casella, quindi ne colora l'interno con il motivo creato nei punti precedenti:

```
SCREEN 8
DIM Riga$(1 TO 4)

' Due linee alternate magenta e gialle:
Riga$(1) = CHR$(&HC3) + CHR$(&H3C) + CHR$(&HFF) + CHR$(&H3C)
Riga$(2) = Riga$(1)

' Motivo inverso (due linee alternate gialle e magenta):
Riga$(3) = CHR$(&H3C) + CHR$(&HC3) + CHR$(&HFF) + CHR$(&HC3)
Riga$(4) = Riga$(3)

' Crea una casella motivo dalle righe così definite:
FOR I% = 1 TO 4
   Casella$ = Casella$ + Riga$(I%)
NEXT I%

' Traccia un rettangolo e lo riempie con il motivo:
LINE (50, 50)-(570, 150), , B
PAINT (320, 100), Casella$
```

5.9 DRAW: un macro linguaggio grafico

L'istruzione **DRAW** è da sola un mini linguaggio. Essa disegna e colora immagini sullo schermo utilizzando un insieme di comandi di una o due lettere, noti come "macro", incorporati in un'espressione a stringa.

DRAW offre i seguenti vantaggi rispetto alle istruzioni grafiche trattate finora:

- L'argomento a stringa macro di **DRAW** è compatto: una singola breve stringa produce lo stesso output di diverse istruzioni **LINE**.
- Le immagini create con **DRAW** possono essere proporzionate facilmente, cioè si possono ingrandire o ridurre inserendo la macro **S** nella stringa macro.
- Le immagini create con **DRAW** possono essere ruotate un numero qualunque di gradi utilizzando la macro **TA** della macro stringa.

Per ulteriori informazioni consultare il Consulente QB.

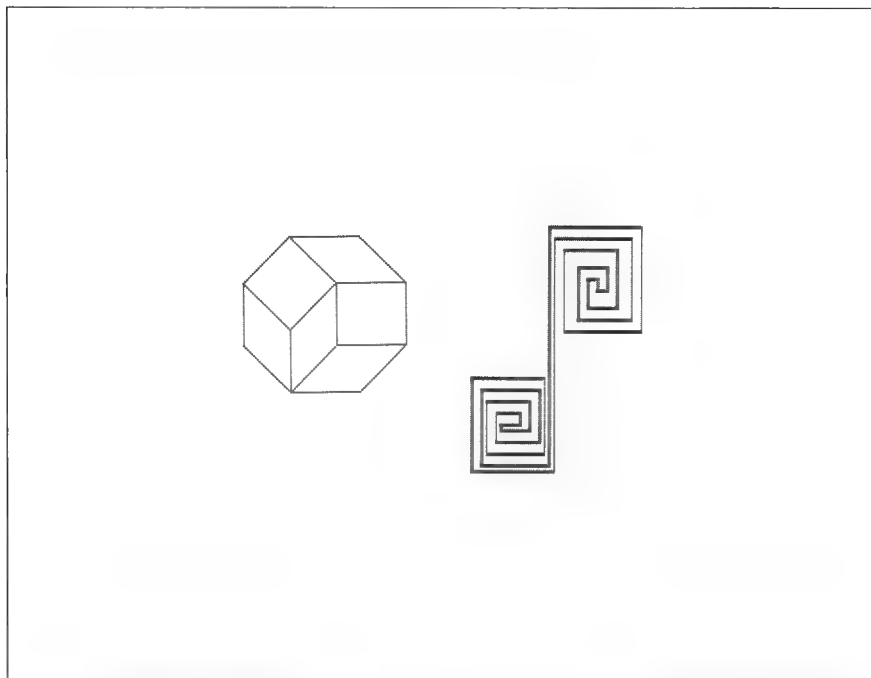
Esempio

Il programma seguente presenta una breve introduzione alle macro di movimento **U**, **D**, **L**, **R**, **E**, **F**, **G**, ed **H**; la macro "tracciare/non tracciare" **B**; e la macro colore **C**. Questo programma disegna linee orizzontali, verticali e diagonali in colori diversi, a seconda del tasto di DIREZIONE premuto sul tastierino numerico (SU, GIU', SINISTRA, PGSU, PGGIU', e così via). Questo programma si trova nel file PLOTTER.BAS sui dischi originali del QuickBASIC.

```
' Valori dei tasti sul tastierino numerico e della BARRA
' SPAZIATRICE:
CONST SU = 72, GIU = 80, SINIS = 75, DES = 77
CONST SUSINIS = 71, SUDES = 73, GIUSINIS = 79, GIUDES = 81
CONST BARRASPAZ = " "

' Null$ è il primo carattere dei due restituiti da INKEY$
' alla lettura dei tasti di direzione come SU e GIU':
Null$ = CHR$(0)

' Se Traccia$ = "" traccia linee; se Traccia$ = "B" sposta
' il cursore grafico, ma non traccia linee:
Traccia$ = ""

PRINT "Usare i tasti di DIREZIONE per tracciare linee."
```

```
PRINT "Premere la BARRA SPAZIATRICE per accendere e";
PRINT "spegnere il disegno delle linee."
PRINT "Premere INVIO per iniziare. Premere u per uscire";
PRINT "dal programma."
DO: LOOP WHILE INKEY$ = ""

SCREEN 1
CLS

DO
   SELECT CASE ValTasto$
      CASE Null$ + CHR$(SU)
         DRAW Traccia$ + "C1 U2"
      CASE Null$ + CHR$(GIU)
         DRAW Traccia$ + "C1 D2"
      CASE Null$ + CHR$(SINIS)
         DRAW Traccia$ + "C2 L2"
      CASE Null$ + CHR$(DES)
         DRAW Traccia$ + "C2 R2"
      CASE Null$ + CHR$(SUSINIS)
         DRAW Traccia$ + "C3 H2"
      CASE Null$ + CHR$(SUDES)
         DRAW Traccia$ + "C3 E2"
      CASE Null$ + CHR$(GIUSINIS)
         DRAW Traccia$ + "C3 G2"
      CASE Null$ + CHR$(GIUDES)
         DRAW Traccia$ + "C3 F2"
      CASE BARRASPAZ
         IF Traccia$ = "" THEN Traccia$ = "B " ELSE _
            Traccia$ = ""
      CASE ELSE
         ' L'utente ha premuto un tasto che non è un tasto
         ' di direzione, né la BARRA SPAZIATRICE, né "u",
         ' quindi rimane inattivo.
   END SELECT

   ValTasto$ = INKEY$

LOOP UNTIL ValTasto$ = "u"

SCREEN 0, 0
WIDTH 80
END
```

Output

Ecco un esempio campione creato con questo programma:

5.10 Tecniche fondamentali di animazione

Con le sole istruzioni grafiche descritte nelle sezioni precedenti, si possono animare semplici oggetti sullo schermo. Per esempio, si può disegnare un cerchio con **CIRCLE**, poi cancellarlo ridisegnandolo con il colore di sfondo, e poi ricalcolare il centro del cerchio e disegnarlo in un altro posto.

Questa tecnica è abbastanza efficace per le figure semplici, ma quando si creano immagini più complesse presenta notevoli svantaggi. Pur essendo veloci, le istruzioni grafiche non sono istantanee, e si noteranno ritardi; e poi se si cancella un oggetto dallo schermo vengono cancellate anche altre cose presenti sullo schermo.

GET e **PUT** permettono invece un'animazione molto veloce. **GET** genera una copia di un'immagine creata con un'istruzione **PSET**, **LINE**, **CIRCLE**, o **PAINT**, e la carica in memoria. **PUT** poi riproduce l'immagine in qualunque parte della finestra o dello schermo.

5.10.1 Memorizzazione delle immagini con GET

Dopo aver creato sullo schermo l'immagine originale, calcolare le coordinate *x* e *y* di un rettangolo abbastanza grande da contenere l'intera immagine. Utilizzare quindi **GET** per copiare in memoria l'intero rettangolo. La sintassi per l'istruzione grafica **GET** è

GET [STEP] (*x1*, *y1*)**–[STEP]** (*x2*, *y2*)**,** *nomematrice*

dove (*x1*, *y1*) e (*x2*, *y2*) danno le coordinate degli angoli superiore sinistro e inferiore destro di un rettangolo. L'argomento *nomematrice* si riferisce a una matrice numerica. La dimensione da specificare per *nomematrice* in un'istruzione **DIM** dipende dai tre fattori seguenti:

- L'altezza e la larghezza del rettangolo che racchiude l'immagine
- La modalità schermo selezionata per l'output grafico
- Il tipo della matrice (interi, interi lunghi, in precisione semplice o doppia)

Nota Sebbene la matrice utilizzata per memorizzare le immagini possa contenere qualunque tipo numerico, si consiglia vivamente di utilizzare soltanto matrici a interi. Tutti i motivi grafici possibili sullo schermo possono essere rappresentati da interi. Non è così per i numeri reali, in precisione semplice o doppia: alcuni motivi grafici non sono numeri reali validi, e la memorizzazione di questi motivi in una matrice di numeri reali potrebbe condurre a risultati imprevisti.

La formula per calcolare la dimensione in byte di *nomematrice* è

dimensione-in-byte = 4 + *altezza* * *strati* * **INT**((*larghezza* * *bit-per-pixel* / *strati* + 7) / 8)

dove *altezza* e *larghezza* sono le dimensioni (in numero di pixel) del rettangolo da copiare, e il valore di *bit-per-pixel* dipende dal numero di colori disponibili nella determinata modalità schermo. Più sono i colori e più saranno i bit richiesti per definire ciascun pixel. In modalità schermo 1, due bit definiscono un pixel, mentre in modalità schermo 2 è sufficiente un solo bit. (Per la formula generale relativa ai bit per pixel, consultare la precedente sezione 5.8.2.1, "Dimensioni della casella motivo in diverse modalità schermo".) L'elenco seguente presenta il valore di *strati* per ciascuna modalità schermo:

Modalità schermo	***Numero di strati di bit***
1, 2, 11, e 13	1
9 (64K di memoria video) e 10	2
7, 8, 9 (più di 64K di memoria video), e 12	4

Per ottenere il numero degli elementi necessario nella matrice, dividere *dimensione-in-byte* per il numero di byte in ciascun elemento della matrice. A questo punto entra in gioco il tipo della matrice. Se si tratta di una matrice a interi, ciascun elemento prende due byte di memoria (la dimensione di un intero), e per ottenere la dimensione effettiva della matrice, occorre dividere *dimensione-in-byte* per due. Analogamente, se si tratta di una matrice a interi lunghi, occorre dividere *dimensione-in-byte* per quattro (perché un intero lungo richiede quattro byte di memoria), e così via. Se si tratta di una matrice in precisione semplice, bisogna dividere per quattro; se in precisione doppia, per otto.

Le indicazioni seguenti spiegano il modo per calcolare la dimensione di una matrice intera abbastanza grande da contenere un rettangolo in modalità schermo 1 con coordinate (10, 40) per l'angolo superiore sinistro e (90, 80) per l'angolo inferiore destro:

1. Calcolare l'altezza e la larghezza del rettangolo:

   AltezzaRettangolo = **ABS** (*y2* − *y1*) + 1 = 80 − 40 + 1 = 41
   LarghezzaRettangolo = **ABS** (*x2* − *x1*) + 1 = 90 − 10 + 1 = 81

   Ricordarsi di aggiungere 1 dopo aver sottratto *y1* da *y2* o *x1* da *x2*. Per esempio, se *x1* = 10 e *x2* = 20, la larghezza del rettangolo è 20 − 10 + 1, cioè 11.

2. Calcolare la dimensione in byte della matrice di interi:

   DimensioneinByte = 4 + AltezzaRettangolo * **INT**((LarghezzaRettangolo * BitPerPixel + 7) / 8)
   = 4 + 41 * **INT**((81 * 2 + 7) / 8)
   = 4 + 41 * **INT**(169 / 8)
   = 4 + 41 * 21
   = 865

3. Dividere la dimensione in byte per i byte di ciascun elemento (due per gli interi) e arrotondare il risultato al numero intero più vicino:

   865 / 2 = 433

Di conseguenza se il nome della matrice è `Immagine`, la seguente istruzione DIM assicura che `Immagine` sia abbastanza grande da copiare i dati dei pixel nel rettangolo:

```
DIM Immagine (1 TO 433) AS INTEGER
```

Nota Sebbene l'istruzione **GET** utilizzi coordinate logiche dopo un'istruzione **WINDOW**, per calcolare la dimensione della matrice utilizzata in **GET** è necessario utilizzare le coordinate fisiche. (Per ulteriori informazioni relative a **WINDOW** e al modo in cui convertire le coordinate logiche in coordinate fisiche, consultare la precedente sezione 5.6, "Ridefinizione delle coordinate di una finestra con WINDOW".)

I punti qui sopra delineati forniscono la dimensione minima richiesta per la matrice; qualunque dimensione più grande sarà, comunque, sufficiente. Per esempio, anche l'istruzione seguente si potrebbe usare:

```
DIM Immagine (1 TO 500) AS INTEGER
```

Esempio

Il programma seguente disegna un'ellisse e ne colora l'interno. Un'istruzione **GET** copia in memoria un'area rettangolare che contiene l'ellisse. (La seguente sezione 5.10.2, "Spostamento di immagini con PUT", descrive come utilizzare l'istruzione **PUT** per riprodurre l'ellisse in un punto diverso.)

```
SCREEN 1

' Dimensiona una matrice grande abbastanza da contenere
' il rettangolo:
DIM Immagine (1 TO 433) AS INTEGER

' Traccia un'ellisse dentro il rettangolo, usando un bordo
' magenta e colorandone l'interno bianco:
CIRCLE (50, 60), 40, 2, , , .5
PAINT (50, 60), 3, 2

' Memorizza nella matrice l'immagine del rettangolo:
GET (10, 40)-(90, 80), Immagine
```

5.10.2 Spostamento di immagini con PUT

Mentre l'istruzione **GET** fa la copia di un'immagine, **PUT** la inserisce in una parte qualsiasi dello schermo. Un'istruzione del tipo

PUT (*x*, *y*), *nomematrice* [, *opzione*]

copia l'immagine rettangolare memorizzata in *nomematrice* sullo schermo, collocandone l'angolo superiore sinistro nel punto in cui si trovano le coordinate (*x*, *y*). Si noti che in **PUT** appare soltanto una coppia di coordinate.

Se nel programma appare un'istruzione **WINDOW** prima di **PUT**, le coordinate *x* e *y* saranno relative all'angolo inferiore sinistro del rettangolo. **WINDOW SCREEN**, però, non ha questo effetto; cioè, dopo **WINDOW SCREEN**, *x* e *y* restano relativi all'angolo superiore sinistro del rettangolo.

Aggiungendo la riga riportata di seguito all'ultimo esempio della sezione precedente, si produce sul lato destro dello schermo una copia esatta dell'ellisse colorata, molto più rapidamente che non ridisegnando e ricolorando la stessa figura con **CIRCLE** e **PAINT**:

```
PUT (200, 40), Immagine
```

Bisogna fare attenzione a non specificare delle coordinate che potrebbero portare qualsiasi parte dell'immagine fuori dallo schermo o dalla finestra attiva, come nelle istruzioni seguenti:

```
SCREEN 2
.
.
.
' Il rettangolo misura 141 x 91 pixel:
GET (10, 10)-(150, 100), Immagine
PUT (510, 120), Immagine
```

A differenza di altre istruzioni grafiche quali **LINE** o **CIRCLE**, **PUT** non taglia le parti delle immagini che si trovano al di fuori della finestra. L'istruzione produce invece il messaggio di errore `Chiamata di funzione non ammessa`.

PUT presenta anche il vantaggio di controllare il modo in cui le immagini memorizzate interagiscono con ciò che già si trova sullo schermo, utilizzando l'argomento *opzione*. Questo argomento può essere una delle opzioni seguenti: **PSET**, **PRESET**, **AND**, **OR**, o **XOR**.

Se ciò che si trova già sullo schermo non ha importanza, usare l'opzione **PSET**, che riporta sullo schermo l'esatta copia dell'immagine memorizzata, sovrascrivendo eventuale output preesistente.

La tabella 5.4 mostra l'effetto delle altre opzioni dell'istruzione **PUT** sull'interazione dei pixel dell'immagine memorizzata con quelli già sullo schermo. In questa tabella, 1 indica che un pixel è acceso e 0 che un pixel è spento.

Tabella 5.4 Effetto di opzioni diverse in modalità schermo 2

Opzione	*Pixel nell'immagine memorizzata*	*Pixel sullo schermo prima dell'istruzione PUT*	*Pixel sullo schermo dopo l'istruzione PUT*
PSET	0	0	0
	0	1	0
	1	0	1
	1	1	1
PRESET	0	0	1
	0	1	1
	1	0	0
	1	1	0
AND	0	0	0
	0	1	0
	1	0	0
	1	1	1
OR	0	0	0
	0	1	1
	1	0	1
	1	1	1
XOR	0	0	0
	0	1	1
	1	0	1
	1	1	0

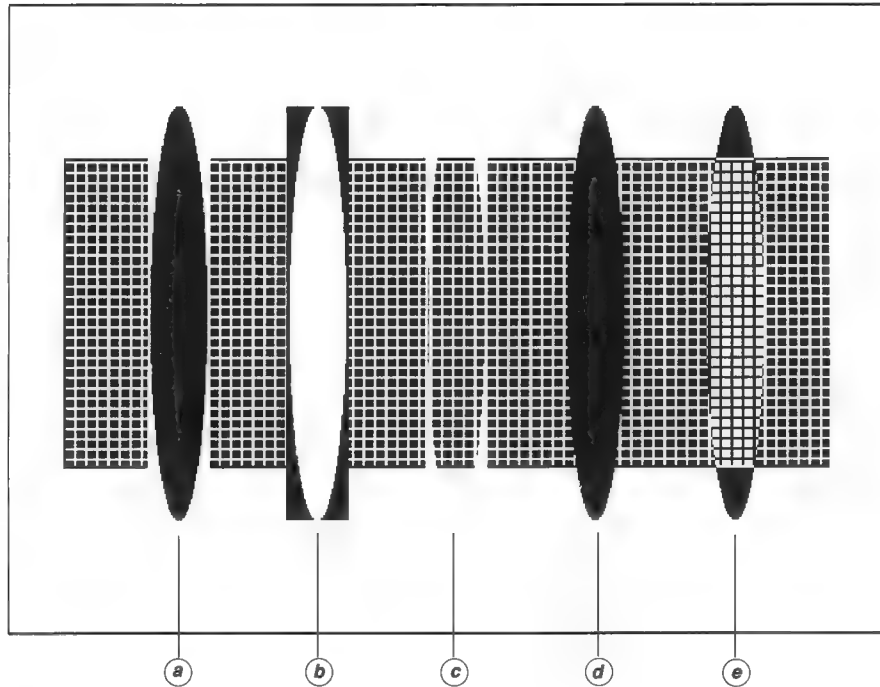
Come si può vedere, queste opzioni fanno sì che un'istruzione **PUT** si comporti con i pixel come gli operatori logici si comportano con i numeri. L'opzione **PRESET**, come l'operatore logico **NOT**, inverte l'immagine memorizzata, a prescindere da ciò che già si trova sullo schermo. Le opzioni **AND**, **OR**, e **XOR** sono identiche agli operatori logici che hanno lo stesso nome; per esempio, l'operazione logica:

```
1 XOR 1
```

dà come risultato 0, proprio come l'opzione **XOR** fa spegnere un pixel quando sia il pixel dell'immagine che quello sullo sfondo sono accesi.

L'output del programma seguente mostra la stessa immagine sovrapposta ad un rettangolo colorato mediante l'utilizzazione di ciascuna delle cinque opzioni descritte qui sopra:

```
SCREEN 2

DIM ImmagCirc (1 TO 485) AS INTEGER
' Traccia e colora un'ellisse e ne memorizza
' l'immagine con GET:
CIRCLE (22, 100), 80, , , , 4
PAINT (22, 100)
GET (0, 20)-(44, 180), ImmagCirc
CLS

' Disegna un riquadro e lo riempie con un motivo:
LINE (40, 40)-(600, 160), , B
Motivo$ = CHR$(126) + CHR$(0) + CHR$(126) + CHR$(126)
PAINT (50, 50), Motivo$

' Colloca le immagini dell'ellisse sul riquadro
' utilizzando opzioni diverse:
PUT (100, 20), ImmagCirc, PSET
PUT (200, 20), ImmagCirc, PRESET
PUT (300, 20), ImmagCirc, AND
PUT (400, 20), ImmagCirc, OR
PUT (500, 20), ImmagCirc, XOR
```

Output

a) PSET
b) PRESET
c) AND
d) OR
e) XOR

Nelle modalità schermo che supportano il colore, le opzioni **PRESET**, **AND**, **OR** e **XOR** producono un'interazione più complessa, perché il rapporto tra pixel accesi e spenti e il relativo colore non è semplice. L'analogia sopra illustrata tra queste opzioni e gli operatori logici permane, tuttavia, anche in queste modalità. Ad esempio, se il pixel corrente sullo schermo è del colore 1 (azzurro in palette 1) e quello nell'immagine sovrapposta è del colore 2 (magenta in palette 1), il colore del pixel ottenuto dopo un'istruzione **PUT** dipenderà dall'opzione scelta, come è illustrato nella tabella 5.5 per soltanto due delle sedici diverse combinazioni di colori dell'immagine e dello sfondo.

Tabella 5.5 Effetto di opzioni diverse sul colore in modalità schermo 1 (palette 1)

Opzione	*Colore del pixel nell'immagine memorizzata*	*Colore del pixel sullo schermo prima dell'istruzione PUT*	*Colore del pixel sullo schermo dopo l'istruzione PUT*
PSET	10 (magenta)	01 (azzurro)	10 (magenta)
PRESET	10 (magenta)	01 (azzurro)	01 (azzurro)
AND	10 (magenta)	01 (azzurro)	00 (nero)
OR	10 (magenta)	01 (azzurro)	11 (bianco)
XOR	10 (magenta)	01 (azzurro)	11 (bianco)

Nella palette 1, all'attributo 1 (01 binario) è assegnato l'azzurro, all'attributo 2 (10 binario) il magenta, e all'attributo 3 (11 binario) il bianco. Per assegnare colori diversi agli attributi 1, 2 e 3, qualora si disponga di una scheda EGA, si può utilizzare l'istruzione **PALETTE**.

5.10.3 Animazione con GET e PUT

Le istruzioni **GET** e **PUT** risultano particolarmente utili per ottenere l'animazione, e le due opzioni che meglio le si adattano sono **XOR** e **PSET**. L'animazione eseguita con **PSET** è più veloce; ma quest'opzione cancella lo sfondo dello schermo, come dimostra l'output del programma precedente. Al contrario, **XOR** procede più lentamente, ma ripristina lo sfondo dello schermo dopo aver spostato l'immagine. L'animazione con **XOR** viene ottenuta in quattro fasi:

1. Collocare l'oggetto sullo schermo con **XOR**.
2. Ricalcolare la nuova posizione dell'oggetto.
3. Ricollocare l'oggetto alla precedente posizione sullo schermo, utilizzando ancora **XOR**, questa volta per cancellare la prima immagine.
4. Ripetere la prima fase, collocando l'oggetto alla nuova posizione.

A partire dalla terza fase lo sfondo rimane inalterato. Si può ridurre lo sfarfallio riducendo al minimo l'intervallo di tempo tra il punto 4 e il punto 1 e assicurandosi che ci sia una pausa sufficiente tra i punti 1 e 3. Se si anima più di un oggetto, ciascuno di essi va elaborato contemporaneamente, fase per fase.

Se non è importante conservare lo sfondo, l'animazione si può ottenere con l'opzione **PSET**, avendo cura di includere l'immagine copiata con l'istruzione **GET** in un rettangolo più grande. Se l'oggetto viene spostato ogni volta una distanza minore delle dimensioni di questo bordo, ogni volta che all'immagine viene assegnata una nuova posizione, il bordo cancellerà ogni traccia della posizione precedente. Questo metodo risulta molto più rapido di quello sopra descritto in cui viene utilizzata l'opzione **XOR**, poiché per spostare un oggetto è sufficiente una sola istruzione **PUT**.

Esempi

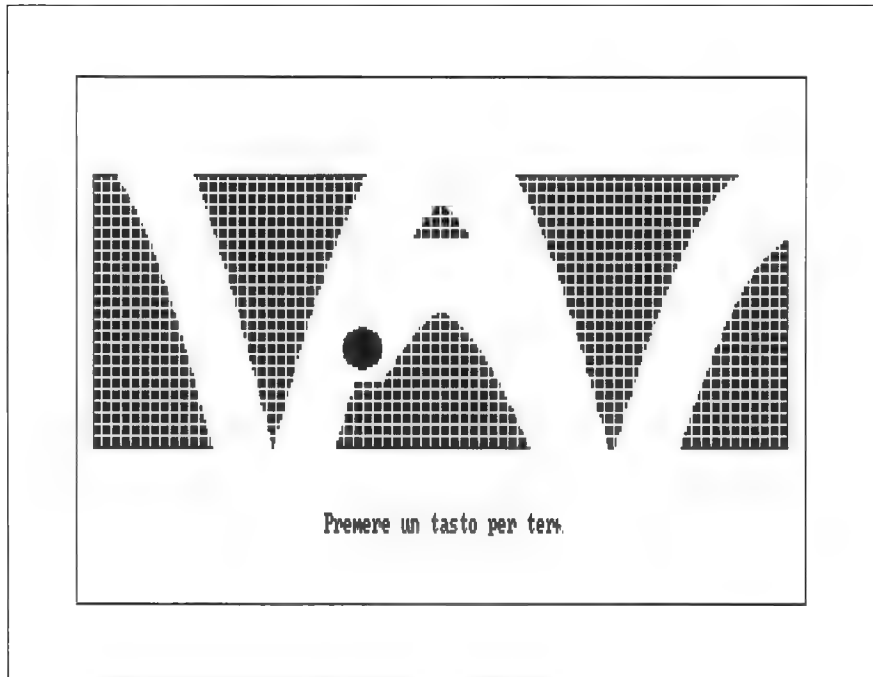
L'esempio seguente mostra come utilizzare **PUT** con l'opzione **PSET** per produrre lo stesso effetto di una palla che rimbalza dal fondo e dai lati di un riquadro. Nell'output che segue si può notare il modo in cui il rettangolo che contiene la palla, specificato nell'istruzione **GET**, cancella anche le caselle colorate e il messaggio visualizzato.

Questo programma si trova nel file RIMBPSET.BAS sui dischi originali del QuickBASIC.

```
DECLARE FUNCTION OttDimMatr (LSin, LDes, LAlto, LBasso)

SCREEN 2
CLS

' Definisce una finestra e la contorna con un bordo:
VIEW (20, 10)-(620, 190), , 1

CONST PIGRECO = 3.141592653589#

' Ridefinisce le coordinate della finestra con coordinate
' logiche:
WINDOW (-3.15, -.14)-(3.56, 1.01)

' Le matrici nel programma ora sono dinamiche:
' $DYNAMIC

' Calcola le coordinate logiche dei lati alto e basso di
' un rettangolo grande abbastanza da contenere l'immagine
' disegnata da CIRCLE e PAINT:
LSin = -.21
LDes = .21
LAlto = .07
LBasso = -.07
```

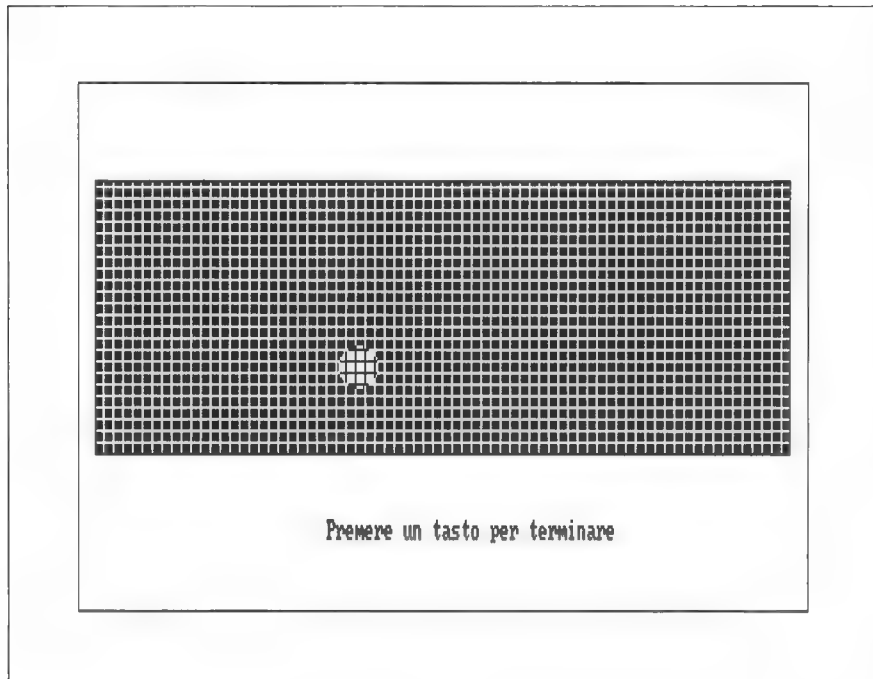
```
' Richiama la funzione OttDimMatr, passandole le coordinate
' logiche del rettangolo:
DimensMatr% = OttDimMatr (LSin, LDes, LAlto, LBasso)

DIM Matrice (1 TO DimensMatr%) AS INTEGER

' Disegna e colora il cerchio:
CIRCLE (0, 0), .18
PAINT (0, 0)

' Memorizza il rettangolo in Matrice:
GET (LSin, LAlto)-(LDes, LBasso), Matrice
CLS

' Disegna un riquadro e lo riempie con un motivo:
LINE (-3, .8)-(3.4, .2), , B
Motivo$ = CHR$(126) + CHR$(0) + CHR$(126) + CHR$(126)
PAINT (0, .5), Motivo$

LOCATE 21, 29
PRINT "Premere un tasto per terminare"

' Inizializza le variabili del ciclo:
Incremento = .02
InizioCiclo = -PIGRECO
AltRimb = 1

DO
   FineCiclo = -InizioCiclo
   FOR X = InizioCiclo TO FineCiclo STEP Incremento

      ' Ogni volta che la palla "rimbalza" (dal fondo della
      ' finestra), la variabile AltRimb decresce, diminuendo
      ' l'altezza del rimbalzo successivo:
      Y = ABS(COS(X)) * AltRimb - .14
      IF Y < -.13 THEN AltRimb = AltRimb * .9

      ' Interrompe se viene premuto un tasto o se AltRimb è
      ' minore di 0,01:
      Esc$ = INKEY$
      IF Esc$ <> "" OR AltRimb < .01 THEN EXIT FOR
```

```
            ' Mette l'immagine sullo schermo. Lo spostamento di
            ' Incremento è minore del bordo intorno al cerchio,
            ' per cui ogni volta che l'immagine si sposta,
            ' cancellerà qualunque traccia lasciata dal precedente
            ' PUT (e qualunque altra cosa occupi quello spazio
            ' sullo schermo):
            PUT (X, Y), Matrice, PSET
        NEXT X

        ' Inverte la direzione:
        Incremento = -Incremento
        InizioCiclo = -InizioCiclo
    LOOP UNTIL Esc$ <> "" OR AltRimb < .01

    Pausa$ = INPUT$(1)
    END

    FUNCTION OttDimMatr (LSin, LDes, LAlto, LBasso) STATIC

        ' Converte le coordinate logiche passate alla funzione
        ' nelle equivalenti coordinate fisiche:
        FSin = PMAP(LSin, 0)
        FDes = PMAP(LDes, 0)
        FAlto = PMAP(LAlto, 1)
        FBasso = PMAP(LBasso, 1)

        ' Calcola l'altezza e la larghezza in pixel del
        ' rettangolo contenente la "palla":
        AltRett = ABS(FBasso - FAlto) + 1
        LargRett = ABS(FDes - FSin) + 1

        ' Calcola le dimensioni in byte della matrice:
        DimensByte = 4 + AltRett * INT((LargRett + 7) / 8)

        ' Matrice è un intero, quindi divide i byte per due:
        OttDimMatr = DimensByte \ 2 + 1
    END FUNCTION
```

Output

Si confronti il programma precedente con il successivo dove lo sfondo dello schermo rimane inalterato grazie all'utilizzo dell'opzione **XOR** nell'istruzione **PUT**. Si noti che il rettangolo che contiene la palla è più piccolo rispetto a quello del programma precedente, perché non è necessario lasciare un bordo. Si noti che occorrono due istruzioni **PUT**: una per rendere visibile l'immagine e l'altra per farla sparire. Infine si osservi il ciclo di ritardo vuoto **FOR...NEXT** tra le istruzioni **PUT**; esso riduce lo sfarfallio che si ha quando l'immagine appare e scompare troppo rapidamente.

Questo programma si trova nel file RIMBXOR.BAS sui dischi originali del QuickBASIC.

```
.
.
.
' Il rettangolo è più piccolo di quello nel programma
' precedente, per cui pure Matrice è più piccola:
LSin = -.18
LDes = .18
LAlto = .05
LBasso = -.05
.
.
.
DO
   FineCiclo = -InizioCiclo
   FOR X = InizioCiclo TO FineCiclo STEP Incremento
      Y = ABS(COS(X)) * AltRimb - .14

      ' La prima istruzione PUT posiziona l'immagine
      ' sullo schermo:
      PUT (X, Y), Matrice, XOR

      ' Un ciclo FOR...NEXT vuoto per ritardare il programma
      ' e ridurre lo sfarfallìo dell'immagine:
      FOR I = 1 TO 5: NEXT I

      IF Y < -.13 THEN AltRimb = AltRimb * .9
      Esc$ = INKEY$
      IF Esc$ <> "" OR AltRimb < .01 THEN EXIT FOR

      ' La seconda istruzione PUT cancella l'immagine e
      ' ripristina lo sfondo:
      PUT (X, Y), Matrice, XOR
   NEXT X

   Incremento = -Incremento
   InizioCiclo = -InizioCiclo
LOOP UNTIL Esc$ <> "" OR AltRimb < .01

END
.
.
.
```

Output

5.10.4 Animazione con pagine di schermo

Questa sezione descrive una tecnica di animazione che utilizza pagine multiple nella memoria video del computer.

Le pagine di una memoria video sono simili a quelle di un libro. A seconda delle capacità grafiche del computer, ciò che si vede visualizzato sullo schermo può essere soltanto una parte di tutta la memoria video disponibile, proprio come accade quando si apre un libro. Tuttavia, a differenza delle pagine di un libro, le pagine non visualizzate possono rimanere attive; per esempio, mentre si sta visualizzando una pagina sullo schermo, output grafici possono essere inviati alle altre.

L'area della memoria video visibile sullo schermo è detta "pagina visiva", mentre quella alla quale è indirizzato l'output delle istruzioni grafiche è detta "pagina attiva". Per selezionare pagine di schermo visive ed attive, occorre utilizzare l'istruzione **SCREEN** con la seguente sintassi:

SCREEN [*modalità*], [, [*pagatt*] [, *pagvis*]]

In questa sintassi, *pagatt* rappresenta il numero della pagina attiva, e *pagvis* il numero della pagina visiva. Esse possono coincidere (e ciò avviene come predefinizione quando gli argomenti *pagatt* e *pagvis* non sono utilizzati con **SCREEN**, nel qual caso il valore di entrambi gli argomenti è 0).

E' possibile animare gli oggetti sullo schermo selezionando una modalità schermo con più pagine di memoria video, e poi alternando le pagine, inviando l'output a una o più pagine attive mentre si visualizza dell'output completato dalla pagina visiva. Questa tecnica rende visiva una pagina attiva solo quando l'output su quella pagina risulta completo. Poiché l'osservatore vede solo un'immagine finita, la visualizzazione è istantanea.

Esempio

Nel programma seguente viene illustrata la tecnica precedente. Viene selezionata la modalità schermo 7, con due pagine; quindi, mediante l'istruzione **DRAW**, si disegna un cubo che viene ruotato di 15° alla volta modificando il valore della macro **TA** nella stringa utilizzata da **DRAW**. Alternando la pagina attiva con la visiva, il programma mostra sempre un cubo finito dalla pagina visiva mentre ne disegna uno nuovo su quella attiva.

Questo programma si trova nel file CUBO.BAS sui dischi originali di QuickBASIC.

```
' La stringa macro che disegna il cubo e ne colora i lati:
Traccia$ = "BR30 BU25 C1 R54 U45 L54 D45 BE20 P1,1 G20 C2 _
   G20" + "R54 E20 L54 BD5 P2,2 U5 C4 G20 U45 E20 D45 BL5 _
   P4,4"

PagAtt% = 1       ' Inizializza i valori della pagina attiva
PagVis% = 0       ' e visiva e dell'angolo di rotazione.
Angolo% = 0

DO

   ' Disegna alla pagina attiva mentre visualizza
   ' la pagina visiva:
   SCREEN 7, , PagAtt%, PagVis%
   CLS 1

   ' Ruota il cubo di "Angolo%" gradi:
   DRAW "TA" + STR$(Angolo%) + Traccia$
```

```
    ' Angolo% è un multiplo di 15 gradi:
    Angolo% = (Angolo% + 15) MOD 360

    ' Scambia la pagina attiva con quella visiva:
    SWAP PagAtt%, PagVis%

LOOP WHILE INKEY$ = ""      ' La pressione di un tasto
                            ' termina il programma.

END
```

5.11 Programmi esempio

I programmi esempio presentati in questo capitolo sono un generatore di istogrammi, un programma che visualizza l'insieme di Mandelbrot utilizzando diversi colori, e un editor di motivi.

5.11.1 Generatore di istogrammi (ISTOGRAM.BAS)

Per disegnare un istogramma a barre piene, il programma utilizza tutte le forme dell'istruzione **LINE** presentate nelle sezioni 5.3.2.1–5.3.2.3. Ciascuna barra è colorata con un motivo specificato dall'istruzione **PAINT**. I dati da inserire nel programma sono le intestazioni, le etichette per gli assi *x* ed *y* ed un gruppo di massimo cinque etichette con i relativi valori per le barre.

Istruzioni e funzioni utilizzate

Il programma mostra l'utilizzo delle seguenti istruzioni grafiche:

- **LINE**
- **PAINT** (con un motivo)
- **SCREEN**

Listato del programma

Di seguito viene listato il programma ISTOGRAM.BAS:

```
' Definisce il tipo per i titoli:
TYPE TipoTitolo
   TitoloPrinc AS STRING * 40
   TitoloX AS STRING * 40
   TitoloY AS STRING * 18
END TYPE

DECLARE SUB TitoliInput (T AS TipoTitolo)
DECLARE FUNCTION TracciaGrafo$ (T AS TipoTitolo, Etich$(), _
   Valore!(), N%)
DECLARE FUNCTION DatiInput% (Etich$(), Valore!())

' Dichiarazione delle variabili per i titoli e i dati
' delle barre:
DIM Titoli AS TipoTitolo, Etich$(1 TO 5), Valore(1 TO 5)

CONST FALSO = 0, VERO = NOT FALSO

DO
   TitoliInput Titoli
   N% = DatiInput%(Etich$(), Valore())
   IF N% <> FALSO THEN
      NuovoGrafo$ = TracciaGrafo$(Titoli, Etich$(), _
         Valore(), N%)
   END IF
LOOP WHILE NuovoGrafo$ = "S"

END

' ======================= DatiInput ==========================
'  Ottiene l'input per le etichette e i valori delle barre.
' ============================================================

FUNCTION DatiInput%(Etich$(), Valore()) STATIC

   ' Inizializza il numero dei valori:
   NumDati% = 0

   ' Visualizza le istruzioni per l'immissione dei dati:
   CLS
```

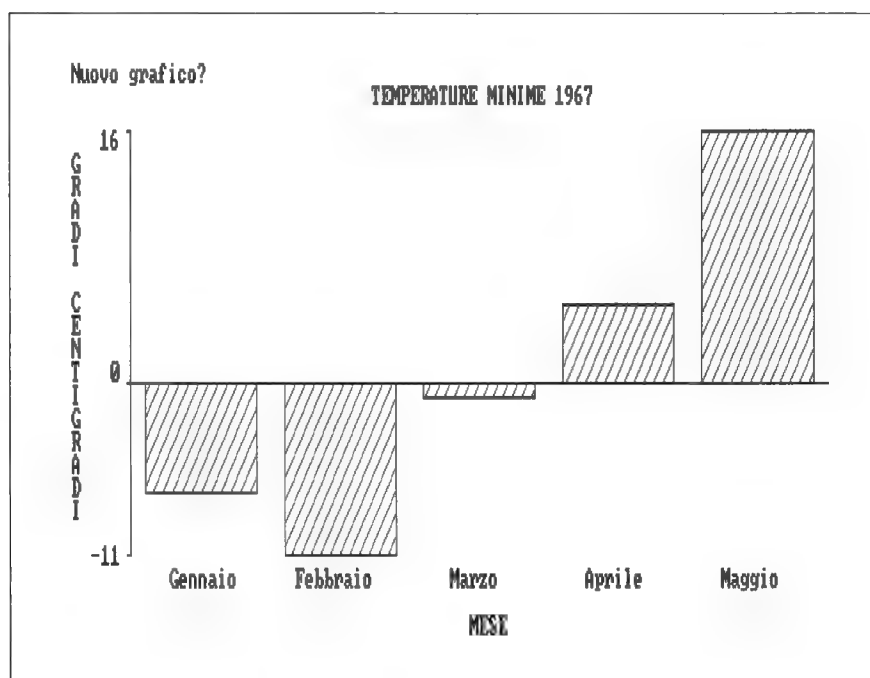
```
    PRINT "Digitare i dati per non più di cinque barre:"
    PRINT "    * Digitare l'etichetta e il valore per ogni ";
    PRINT "barra."
    PRINT "    * I valori possono essere negativi."
    PRINT "    * Immettere un'etichetta vuota per terminare."
    PRINT
    PRINT "Dopo aver visionato il grafico, premere un ";
    PRINT "tasto per terminare il programma."

    ' Smette di leggere dopo 5 dati, oppure se Etich è vuota:
    Fatto% = FALSO
    DO
       NumDati% = NumDati% + 1
       PRINT
       PRINT "Barra("; LTRIM$(STR$(NumDati%)); "):"
       INPUT ; "        Etichetta? ", Etich$(NumDati%)

       ' Legge Valore solo se Etich non è vuota:
       IF Etich$(NumDati%) <> "" THEN
          LOCATE , 35
          INPUT "Valore? ", Valore(NumDati%)

       ' Se Etich è vuota, decrementare il conteggio dei dati
       ' e impostare il valore della flag Fatto a VERO:
       ELSE
          NumDati% = NumDati% - 1
          Fatto% = VERO
       END IF
    LOOP UNTIL (NumDati% = 5) OR Fatto%

    ' Restituisce il numero di valori letti:
    DatiInput% = NumDati%

END FUNCTION

' ===================== TitoliInput ========================
'  Accetta l'input per tre diversi titoli del grafico.
' ==========================================================

SUB TitoliInput (T AS TipoTitolo) STATIC

    ' Imposta lo schermo in modalità testo:
    SCREEN 0, 0
```

```
      ' Legge i titoli
      DO
         CLS
         INPUT "Digitare il titolo principale del grafico: _
            ", T.TitoloPrinc
         INPUT "Digitare il titolo dell'ascissa         : _
            ", T.TitoloX
         INPUT "Digitare il titolo dell'ordinata        : _
            ", T.TitoloY

         ' Verifica che i titoli sono giusti:
         LOCATE 7, 1
         PRINT "Va bene? (Sì per continuare, No per";
         PRINT "cambiare)";
         LOCATE , , 1
         OK$ = UCASE$(INPUT$(1))
      LOOP UNTIL OK$ = "S"
   END SUB

   ' ===================== TracciaGrafo =====================
   '  Disegna un istogramma con i dati letti dalle procedure
   '  TitoliInput e DatiInput.
   ' ========================================================

   FUNCTION TracciaGrafo$ (T AS TipoTitolo, Etich$(), _
      Valore(), N%) STATIC

      ' Imposta le dimensioni del grafico:
      CONST GRAFSUP = 24, GRAFINF = 171
      CONST GRAFSIN = 48, GRAFDES = 624
      CONST LUNGY = GRAFINF - GRAFSUP

      ' Calcola i valori massimo e minimo:
      MassY = 0
      MinY = 0
      FOR I% = 1 TO N%
         IF Valore(I%) < MinY THEN MinY = Valore(I%)
         IF Valore(I%) > MassY THEN MassY = Valore(I%)
      NEXT I%

      ' Calcola la larghezza delle barre e lo spazio
      ' intermedio:
      LargBar = (GRAFDES - GRAFSIN) / N%
      SpazBar = .2 * LargBar
      LargBar = LargBar - SpazBar
```

```
SCREEN 2
CLS

' Traccia l'ordinata:
LINE (GRAFSIN, GRAFSUP)-(GRAFSIN, GRAFINF), 1

' Scrive il titolo principale:
Inizio% = 44 - (LEN(RTRIM$(T.TitoloPrinc)) / 2)
LOCATE 2, Inizio%
PRINT RTRIM$(T.TitoloPrinc);

' Annota l'ordinata:
Inizio% = CINT(13 - LEN(RTRIM$(T.TitoloY)) / 2)
FOR I% = 1 TO LEN(RTRIM$(T.TitoloY))
   LOCATE Inizio% + I% - 1, 1
   PRINT MID$(T.TitoloY, I%, 1);
NEXT I%

' Calcola il fattore di scala perché le etichette non
' superino i 4 digit:
IF ABS(MassY) > ABS(MinY) THEN
   Espon = MassY
ELSE
   Espon = MinY
END IF
Espon = CINT(LOG(ABS(Espon) / 100) / LOG(10))
IF Espon < 0 THEN Espon = 0

' Ridimensiona il minimo e il massimo:
FattoreScala = 10 ^ Espon
MassY = CINT(MassY / FattoreScala)
MinY = CINT(MinY / FattoreScala)

' Se l'esponente non è zero, visualizza sul grafico il
' fattore di scala:
IF Espon <> 0 THEN

   LOCATE 3, 2
   PRINT "x 10^"; LTRIM$(STR$(Espon))
END IF

' Mette una tacca e il valore numerico del massimo
' sull'ordinata:
LINE (GRAFSIN - 3, GRAFSUP)-STEP(3, 0)
LOCATE 4, 2
PRINT USING "####"; MassY
```

```
' Mette una tacca e il valore numerico del minimo
' sull'ordinata:
LINE (GRAFSIN - 3, GRAFINF)-STEP(3, 0)
LOCATE 22, 2
PRINT USING "####"; MinY

' Ridimensiona il minimo e il massimo per calcolare il
' display:
MassY = MassY * FattoreScala
MinY = MinY * FattoreScala

' Annota l'ascissa:
Inizio% = 44 - (LEN(RTRIM$(T.TitoloX)) / 2)
LOCATE 25, Inizio%
PRINT RTRIM$(T.TitoloX);

' Calcola l'intervallo dei pixel per l'ordinata:
IntervY = MassY - MinY

' Definisce un motivo a strisce diagonali:
Casella$ = CHR$(1) + CHR$(2) + CHR$(4) + CHR$(8) + _
   CHR$(16) + CHR$(32) + CHR$(64) + CHR$(128)

' Disegna una linea allo zero se è appropriato:
IF MinY < 0 THEN
   Basso = GRAFINF - ((-MinY) / IntervY * LUNGY)
   LOCATE INT((Basso - 1) / 8) + 1, 5
   PRINT "0";
ELSE
   Basso = GRAFINF
END IF

' Disegna l'ascissa:
LINE (GRAFSIN - 3, Basso)-(GRAFDES, Basso)

' Disegna le barre e le etichette:
Inizio% = GRAFSIN + (SpazBar / 2)
FOR I% = 1 TO N%

   ' Disegna una etichetta di barra:
   CentroBarra = Inizio% + (LargBar / 2)
   CentroCaratt = INT((CentroBarra - 1) / 8) + 1
   LOCATE 23, CentroCaratt - INT(LEN(RTRIM$(Etich$ _
      (I%))) / 2)
   PRINT Etich$(I%);
```

```
      ' Disegna la barra e la colora con il motivo a
      ' strisce:
      AltezzaBarra = (Valore(I%) / IntervY) * LUNGY
      LINE (Inizio%, Basso)-STEP(LargBar, -AltezzaBarra),,B
      PAINT (CentroBarra, Basso - (AltezzaBarra / 2)), _
         Casella$, 1

      Inizio% = Inizio% + LargBar + SpazBar
   NEXT I%

   LOCATE 1, 1, 1
   PRINT "Nuovo grafico? ";
   TracciaGrafo$ = UCASE$(INPUT$(1))

END FUNCTION
```

Output

5.11.2 Colore in una figura creata matematicamente (MANDEL.BAS)

E' possibile creare una figura "frattale" mediante le istruzioni grafiche del BASIC. Un frattale è la rappresentazione grafica delle variazioni subite da numeri sottoposti a particolari sequenze infinite di operazioni matematiche. Il frattale generato da questo programma fa parte di un sottoinsieme della classe dei numeri complessi; è denominato "insieme di Mandelbrot" dal nome di Benoit B. Mandelbrot del Thomas J. Watson Research Center dell'IBM.

In breve, i numeri complessi si compongono di due parti: una parte reale e una parte immaginaria, multiplo della radice quadrata di −1. Elevando al quadrato un numero complesso, quindi inserendo la parte reale ed immaginaria in un secondo numero complesso, rielevando al quadrato il numero risultante, e ripetendo di seguito l'operazione, ne risulterà che il valore di alcuni numeri complessi aumenta rapidamente mentre quello di altri oscilla presso un valore costante. I valori stabili fanno parte dell'insieme di Mandelbrot e verranno rappresentati in colore nero, mentre gli altri verranno rappresentati in tutti gli altri colori della palette. Quanto più piccolo è l'attributo colore, tanto più instabile è il punto.

Per approfondire l'argomento riguardante l'insieme di Mandelbrot si consiglia la lettura dell'articolo di A. K. Dewdney, "Computer Recreations", su *Scientific American*, Agosto 1985.

Inoltre, il programma verifica la presenza di una scheda EGA, ed in caso affermativo disegna l'insieme di Mandelbrot in modalità schermo 8. Dopo aver disegnato ogni linea, il programma alterna i 16 colori della palette con l'istruzione **PALETTE USING**. Se non esiste una scheda EGA, il programma presenta l'insieme di Mandelbrot nei 4 colori della modalità schermo 1 (bianco, magenta, azzurro e nero).

Istruzioni e funzioni utilizzate

Questo programma mostra l'utilizzo delle seguenti istruzioni grafiche:

- **LINE**
- **PALETTE USING**
- **PMAP**
- **PSET**
- **SCREEN**
- **VIEW**
- **WINDOW**

Listato del programma

```
DEFINT A-Z        ' Il tipo predefinito di variabile è intero.

DECLARE SUB ScorrePalette()
DECLARE SUB ValFinestra(LS%, LD%, LA%, LB%)
DECLARE SUB ProvaSchermo(ME%, GC%, FS%, FD%, FA%, FB%)

CONST FALSO = 0, VERO = NOT FALSO        ' Costanti booleane

' Imposta il numero massimo di iterazioni per ogni punto:
CONST LIMCICLO = 30, LIMDIMENS = 1000000

DIM MatrPalette(15)
FOR I = 0 TO 15 : MatrPalette(I) = I : NEXT I

' Chiama ValFinestra per stabilire le coordinate logiche
' della finestra:
ValFinestra LSin, LDes, LAlt, LBas

' Chiama ProvaSchermo per stabilire se il computer è dotato
' di EGA, e per ottenere le coordinate fisiche degli angoli
' della finestra:
ProvaSchermo ModEga, GammaColori, FSin, FDes, FAlt, FBas

' Definisce la finestra con le corrispondenti coordinate
' fisiche e logiche:
VIEW (FSin, FAlt)-(FDes, FBas), 0, GammaColori
WINDOW (LSin, LAlt)-(LDes, LBas)

LOCATE 24, 10 : PRINT "Premere un tasto per terminare.";

LungX = FDes - FSin
LungY = FBas - FAlt
LargColori = LIMCICLO \ GammaColori

' Esamina ogni pixel della finestra e calcola se
' appartiene all'insieme di Mandelbrot:
FOR Y = 0 TO LungY           ' Il ciclo esterno esamina ogni
                             ' riga della finestra.
   LogicY = PMAP(Y, 3)       ' Ottiene la coordinata logica y
                             ' del pixel.
   PSET(LSin, LogicY)        ' Traccia il pixel più a sinistra
                             ' della riga.
   ColPrec = 0               ' Comincia col colore di sfondo.
```

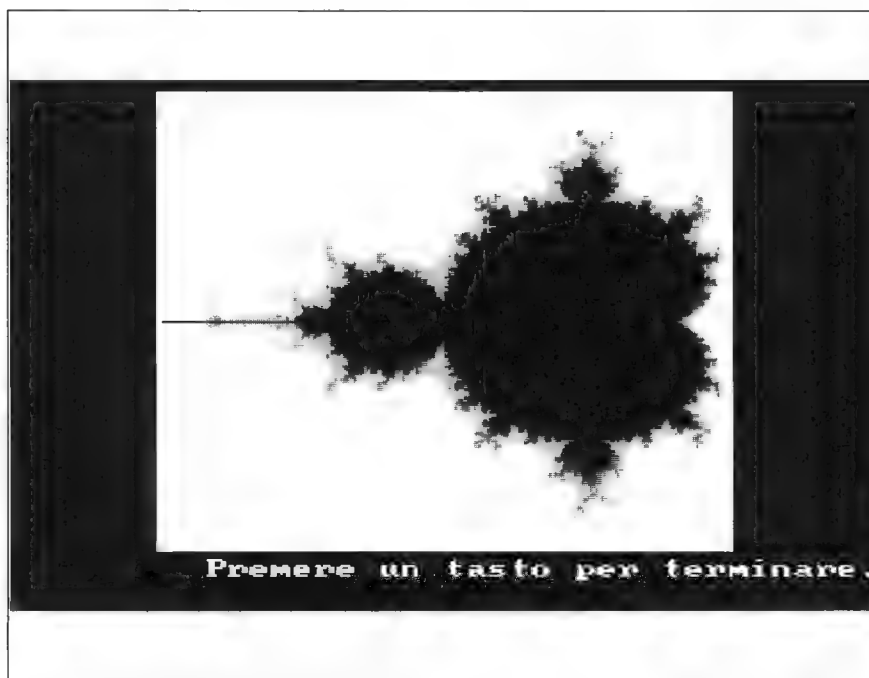
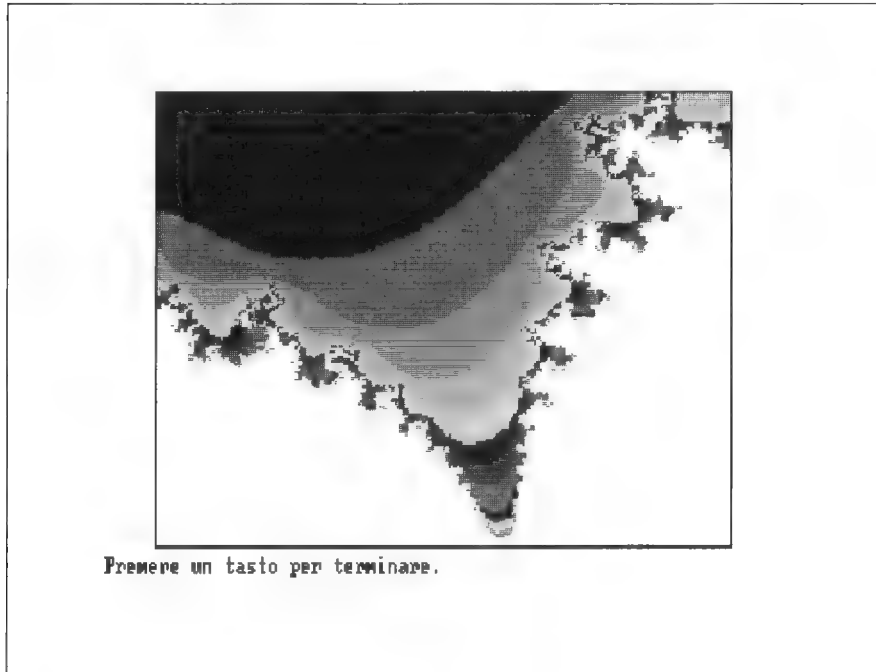
```
      FOR X = 0 TO LungX       ' Questo ciclo esamina ogni pixel
                               ' della riga.
         LogicX = PMAP(X, 2)   ' Ottiene la coordinata logica x
                               ' del pixel.
         MandelX& = LogicX
         MandelY& = LogicY

         ' Calcola se il punto appartiene all'insieme di
         ' Mandelbrot:
         FOR I = 1 TO LIMCICLO
            NumReal& = MandelX& * MandelX&
            NumImag& = MandelY& * MandelY&
            IF (NumReal& + NumImag&) >= LIMDIMENS THEN EXIT FOR
            MandelY& = (MandelX& * MandelY&) \ 250 + LogicY
            MandelX& = (NumReal& - NumImag&) \ 500 + LogicX
         NEXT I

         ' Assegna un colore al punto:
         ColPunto = I \ LargColori

         ' Se il colore è cambiato, traccia una linea dal
         ' precedente punto di riferimento al nuovo punto,
         ' usando il precedente colore:
         IF ColPunto <> ColPrec THEN
            LINE -(LogicX, LogicY), (GammaColori - ColPrec)
            ColPrec = ColPunto
         END IF

         IF INKEY$ <> "" THEN END
      NEXT X

      ' Traccia l'ultimo segmento fino al bordo destro della
      ' finestra:
      LINE -(LogicX, LogicY), (GammaColori - ColPrec)

      ' Se il computer è dotato di EGA, fa scorrere la palette
      ' dopo aver tracciato il segmento:
      IF ModEga THEN ScorrePalette
   NEXT Y

   DO
      ' Continua a far scorrere la palette finché l'utente non
      ' preme un tasto:
      IF ModEga THEN ScorrePalette
   LOOP WHILE INKEY$ = ""
```

```
SCREEN 0, 0                ' Ripristina lo schermo in modalità
WIDTH 80                   ' testo a 80 colonne.
END

SchermoInf:                ' Gestore di errori chiamato se il
   ModEga = FALSO          ' computer non è dotato di scheda
                           ' grafica EGA.
   RESUME NEXT

' ===================== ProvaSchermo =======================
'  Con il comando SCREEN 0, controlla se il computer è
'  dotato di EGA. Se questo causa un errore, il flag ME
'  viene impostato su FALSO e lo schermo viene impostato
'  con SCREEN 1.
'
'  Imposta poi i valori delle coordinate fisiche degli
'  angoli della finestra (FS = sinistro, FD = destro,
'  FA = alto, FB = basso), tenendo conto del rapporto
'  delle dimensioni per ottenere una finestra quadrata.
' ==========================================================

SUB ProvaSchermo (ME, GC, FS, FD, FA, FB) STATIC
   ME = VERO
   ON ERROR GOTO SchermoInf
   SCREEN 8, 1
   ON ERROR GOTO 0

   IF ME THEN              ' Nessun errore, SCREEN 8 va bene.
      FS = 110 : FD = 529
      FA = 5 : FB = 179
      GC = 15              ' 16 colori (0-15)

   ELSE                    ' Errore, usare SCREEN 1.
      SCREEN 1, 1
      FS = 55 : FD = 264
      FA = 5 : FB = 179
      GC = 3               ' 4 colori (0-3)
   END IF

END SUB

' ===================== ScorrePalette ======================
'  Fa scorrere la palette di un colore ogni chiamata.
' ==========================================================

SUB ScorrePalette STATIC
   SHARED MatrPalette(), GammaColori
```

```
   FOR I = 1 TO GammaColori
      MatrPalette(I) = (MatrPalette(I) MOD GammaColori) + 1
   NEXT I
   PALETTE USING MatrPalette(0)

END SUB

' ===================== ValFinestra ========================
'  Legge le coordinate logiche degli angoli della finestra
'  digitate dall'utente, o imposta i valori predefiniti in
'  mancanza di input.
' ==========================================================

SUB ValFinestra (LS, LD, LA, LB) STATIC
   CLS
   PRINT "Questo programma visualizza una immagine grafica "
   PRINT "dell'intero insieme di Mandelbrot. La finestra "
   PRINT "predefinita va da (-1000, 625) a (250, -625). Per "
   PRINT "vedere un ingrandimento di una parte del "
   PRINT "grafico, digitarne le rispettive coordinate. "
   PRINT
   PRINT "Premere <INVIO> per usare la finestra "
   PRINT "predefinita. Premere un altro tasto per "
   PRINT "impostare coordinate diverse: ";
   LOCATE , , 1
   Risp$ = INPUT$(1)

   ' L'utente non ha premuto <INVIO>, quindi legge le
   ' coordinate logiche della finestra:
   IF Risp$ <> CHR$(13) THEN
      PRINT
      INPUT "Coordinata X dell'angolo in alto a sinistra: _
         ", LS
      DO
         INPUT "Coordinata X dell'angolo in basso a destra: _
            ", LD
         IF LD <= LS THEN
         PRINT "L'angolo destro deve essere maggiore di ";
         PRINT "quello sinistro."
         END IF
      LOOP WHILE LD <= LS
      INPUT "Coordinata Y dell'angolo in alto a sinistra:  _
         ", LA
      DO
         INPUT "Coordinata Y dell'angolo in basso a destra: _
            ", LB
```

```
         IF LB >= LA THEN
         PRINT "L'angolo in basso deve essere minore di ";
         PRINT "quello in alto."
         END IF
      LOOP WHILE LB >= LA

   ELSE          ' L'utente ha premuto <INVIO>; imposta i
                 ' valori predefiniti.
      LS = -1000
      LD = 250
      LA = 625
      LB = -625
   END IF
END SUB
```

Output

L'illustrazione seguente presenta l'insieme di Mandelbrot in modalità schermo 1. Questo è l'output che si ottiene con una scheda CGA scegliendo le coordinate predefinite per la finestra.

L'illustrazione seguente mostra l'insieme di Mandelbrot con coordinate (−500, 250) per l'angolo superiore sinistro e (−300, 50) per quello inferiore destro. Questa illustrazione viene rappresentata in modalità schermo 8, quella predefinita per EGA o VGA.

5.11.3 Editor di motivi (EDMOT.BAS)

Questo programma consente di modificare una casella motivo con l'istruzione **PAINT**. Mentre si modifica la casella sul lato sinistro dello schermo, si può controllare il risultante motivo visualizzandolo sul lato destro. Una volta finite le modifiche, il programma visualizza gli argomenti interi con cui l'istruzione **CHR$** ha disegnato ciascuna riga della casella.

Istruzioni e funzioni utilizzate

Il programma mostra l'utilizzo delle seguenti istruzioni grafiche:

- **LINE**
- **PAINT** (con motivo)
- **VIEW**

Listato del programma

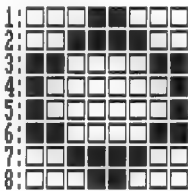
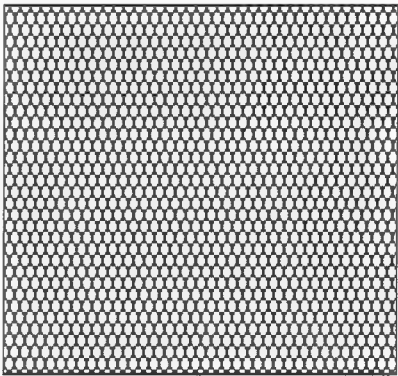
```
DECLARE SUB DisegnaMotivo()
DECLARE SUB ModificaMotivo()
DECLARE SUB Inizializza()
DECLARE SUB MostraMotivo(OK$)

DIM Bit%(0 TO 7), Motivo$, Esc$, DimensMotivo%

DO
   Inizializza
   ModificaMotivo
   MostraMotivo OK$
LOOP WHILE OK$ = "S"

END

' ====================== DisegnaMotivo =====================
'  Disegna un rettangolo contenente un motivo sulla destra
'  dello schermo.
' ==========================================================

SUB DisegnaMotivo STATIC
SHARED Motivo$

   VIEW (320, 24)-(622, 150), 0, 1  ' Visualizza solo il
                                    ' rettangolo.
   PAINT (1, 1), Motivo$            ' Lo colora con PAINT.
   VIEW                             ' Visualizza l'intero
                                    ' schermo.

END SUB

' ====================== Inizializza =======================
'  Imposta lo schermo e il motivo di partenza.
' ==========================================================

SUB Inizializza STATIC
SHARED Motivo$, Esc$, Bit%(), DimensMotivo%

   Esc$ = CHR$(27)            ' Il carattere ESC è ASCII 27.
```

```
' Imposta una matrice contenente bit nelle posizioni da
' 0 a 7:
FOR I% = 0 TO 7
   Bit%(I%) = 2 ^ I%
NEXT I%

CLS

' Legge le dimensioni del motivo (in numero di byte):
LOCATE 5, 5
PRINT "Digitare le dimensioni del motivo (1-16 righe):";
DO
   LOCATE 5, 53
   PRINT "    ";
   LOCATE 5, 53
   INPUT "", DimensMotivo%
LOOP WHILE DimensMotivo% < 1 OR DimensMotivo% > 16
' Accende tutti i bit come motivo iniziale:
Motivo$ = STRING$(DimensMotivo%, 255)

SCREEN 2                      ' Modalità grafica
                              ' monocromatica 640 x 200.

' Traccia le linee divisorie:
LINE (0, 10)-(635, 10), 1
LINE (300, 0)-(300, 152)
LINE (302, 0)-(302, 152)

' Stampa i titoli:
LOCATE 1, 11: PRINT "Byte del motivo"
LOCATE 1, 47: PRINT "Visualizzazione del motivo"

' Traccia la finestra per modifiche al motivo:
FOR I% = 1 TO DimensMotivo%

   ' Stampa l'etichetta alla sinistra di ogni riga:
   LOCATE I% + 3, 8
   PRINT USING "##:"; I%

   ' Disegna le caselle dei "bit":
   X% = 80
   Y% = (I% + 2) * 8
```

```
      FOR J% = 1 TO 8
         LINE (X%, Y%)-STEP(13, 6), 1, BF
         X% = X% + 16
      NEXT J%
   NEXT I%

   DisegnaMotivo                 ' Traccia il rettangolo per la
                                 ' visualizzazione del motivo.
   LOCATE 21, 1
   PRINT "I tasti di DIREZIONE..........Spostano il cursore"
   PRINT "La BARRA SPAZIATRICE.............Cambia il colore"
   PRINT "Il tasto INVIO..............Visualizza il motivo"
   PRINT "Il tasto ESC................Termina la sessione";

END SUB

' ===================== ModificaMotivo =====================
'  Modifica un motivo di caselle-byte.
' ==========================================================

SUB ModificaMotivo STATIC
SHARED Motivo$, Esc$, Bit%(), DimensMotivo%

   NumByte% = 1                  ' Posizione di partenza.
   NumBit% = 7
   Null$ = CHR$(0)               ' CHR$(0) è il primo byte
                                 ' della stringa a due byte
                                 ' restituita quando viene
                                 ' premuto un tasto di
                                 ' DIREZIONE come SU o GIU'.
   DO

      ' Calcola la posizione di partenza sullo schermo di
      ' questo bit:
      X% = ((7 - NumBit%) * 16) + 80
      Y% = (NumByte% + 2) * 8

      ' Aspetta la pressione di un tasto (e fa lampeggiare
      ' il cursore ogni 3/10 di secondo):
      Stato% = 0
      TempoRif = 0
      DO
```

```
      ' Controlla il temporizzatore e cambia lo stato del
      ' cursore se sono passati 3/10 di secondo:
      IF ABS(TIMER - TempoRif) > .3 THEN
         TempoRif = TIMER
         Stato% = 1 - Stato%

         ' Accende e spegne il bordo del bit:
         LINE (X% - 1, Y% - 1)-STEP(15, 8), Stato%, B
      END IF

      Controlla$ = INKEY$         ' Controlla la pressione
                                  ' di un tasto.

   LOOP WHILE Controlla$ = ""     ' Cicla finché non viene
                                  ' premuto un tasto.

   ' Cancella il cursore:
   LINE (X% - 1, Y% - 1)-STEP(15, 8), 0, B

   ' Risponde alla pressione di un tasto:
   SELECT CASE Controlla$

      CASE CHR$(27)
         ' E' stato premuto il tasto ESC: esce da questo
         ' sottoprogramma.
         EXIT SUB

      CASE CHR$(32)
      ' E' stata premuta la BARRA SPAZIATRICE: reimposta
      ' lo stato del bit.

         ' Inverte il bit nella stringa di motivo:
         ByteCorrente% = ASC(MID$(Motivo$, NumByte%, 1))
         ByteCorrente% = ByteCorrente% XOR Bit%(NumBit%)
         MID$(Motivo$, NumByte%) = CHR$(ByteCorrente%)

         ' Ridisegna il bit sullo schermo:
         IF (ByteCorrente% AND Bit%(NumBit%)) <> 0 THEN
            ColoreCorrente% = 1
         ELSE
            ColoreCorrente% = 0
         END IF
         LINE (X% + 1, Y% + 1)-STEP(11, 4), _
            ColoreCorrente%, BF
```

```
         CASE CHR$(13)
         ' E' stato premuto il tasto INVIO: disegna il
         ' motivo nel riquadro a destra.
            DisegnaMotivo

         CASE Null$ + CHR$(75)
         ' E' stato premuto il tasto SINISTRA: sposta il
         ' cursore a sinistra.
            NumBit% = NumBit% + 1
            IF NumBit% > 7 THEN NumBit% = 0

         CASE Null$ + CHR$(77)
         ' E' stato premuto il tasto DESTRA: sposta il
         ' cursore a destra.
            NumBit% = NumBit% - 1
            IF NumBit% < 0 THEN NumBit% = 7

         CASE Null$ + CHR$(72)
         ' E' stato premuto il tasto SU: sposta il cursore
         ' in alto.
            NumByte% = NumByte% - 1
            IF NumByte% < 1 THEN NumByte% = DimensMotivo%

         CASE Null$ + CHR$(80)
         ' E' stato premuto il tasto GIU': sposta il cursore
         ' in basso.
            NumByte% = NumByte% + 1
            IF NumByte% > DimensMotivo% THEN NumByte% = 1

         CASE ELSE
         ' L'utente ha premuto un tasto diverso da ESC,
         ' BARRA SPAZIATRICE, INVIO, SU, GIU', SINISTRA,
         ' o DESTRA; non fare nulla.
      END SELECT
   LOOP
END SUB

' ===================== MostraMotivo =======================
'  Stampa i valori di CHR$ usati da PAINT per costruire
'  il motivo.
' ==========================================================

SUB MostraMotivo(OK$) STATIC
SHARED Motivo$, DimensMotivo%
```

```
   ' Riporta lo schermo in modalità testo a 80 colonne:
   SCREEN 0, 0
   WIDTH 80

   PRINT "Il motivo è composto dai caratteri seguenti:"
   PRINT

   ' Stampa il valore di ogni byte del motivo:
   FOR I% = 1 TO DimensMotivo%
      ByteMotivo% = ASC(MID$(Motivo$, I%, 1))
      PRINT "CHR$("; LTRIM$(STR$(ByteMotivo%)); ")"
   NEXT I%

   PRINT
   LOCATE , , 1
   PRINT "Un altro motivo? (Sì/No)";
   OK$ = UCASE$(INPUT$(1))
END SUB
```

Output

Questo è un esempio di motivo creato dall'editor.

6 Gestione degli errori e degli eventi

Questo capitolo riguarda la gestione degli errori ed eventi che si verificano durante l'esecuzione di un programma. La gestione degli errori evita che il programma si arresti per esempio all'immissione di una stringa dove è richiesto un valore numerico, o al tentativo di aprire un file in una directory non esistente. Mediante la gestione degli eventi, invece, il programma può rilevare e rispondere in tempo reale ad eventi quali pressioni di tasto o dati in arrivo ad una porta **COM**.

Alla fine di questo capitolo, si sapranno eseguire i processi seguenti:

- Attivare la gestione degli errori o degli eventi
- Scrivere una routine per gestire gli errori o gli eventi rilevati
- Riprendere il controllo da una routine di gestione degli errori o degli eventi
- Scrivere un programma che risponde alla pressione di qualunque tasto o combinazione di tasti
- Gestire gli errori o gli eventi all'interno di procedure **SUB** o **FUNCTION**
- Gestire gli errori o gli eventi in programmi a più moduli

6.1 Gestione degli errori

La gestione degli errori consente di rilevare gli errori nel corso dell'esecuzione prima che essi causino l'arresto del programma. In mancanza di gestione, se si verificano errori durante l'esecuzione del programma, per esempio il tentativo di aprire un file di dati non esistente, il BASIC visualizza il relativo messaggio di errore e arresta il programma. Se si tratta di una versione autonoma del programma, bisognerà riavviarlo, perdendo i dati digitati o i calcoli eseguiti prima che si verificasse l'errore.

Quando la gestione degli errori è attiva, l'errore rilevato fa saltare il programma ad una "routine di gestione degli errori" scritta dall'utente, che corregge l'errore stesso. Se è possibile prevedere gli errori che possono verificarsi quando qualcun'altro utilizza il programma, la gestione degli errori permetterà di trattare quegli errori in maniera "amichevole".

Le sezioni 6.1.1 e 6.1.2 spiegano come attivare la gestione degli errori, come scrivere una routine per gestire gli errori rilevati, e come riprendere il controllo da tale routine dopo che questa ha gestito l'errore.

6.1.1 Attivazione della gestione degli errori

La gestione degli errori viene attivata per mezzo dell'istruzione **ON ERROR GOTO** *riga*, dove *riga* è un numero o un'etichetta di riga che identifica la prima riga di una routine di gestione degli errori. Dopo che il BASIC ha incontrato un'istruzione **ON ERROR GOTO** *riga*, qualunque errore verificatosi nel corso dell'esecuzione all'interno del modulo contenente tale istruzione causa il salto alla *riga* specificata. (Se nel modulo non esistesse il numero o l'etichetta, apparirebbe il messaggio di errore di esecuzione `Etichetta non definita`.)

E' necessario che venga eseguita un'istruzione **ON ERROR GOTO** perché abbia effetto la gestione degli errori. Di conseguenza, è necessario che il programma incontri l'istruzione **ON ERROR GOTO** prima che una condizione particolare la richieda. Generalmente essa è una delle prime istruzioni eseguibili del modulo principale o di una procedura.

L'istruzione **ON ERROR GOTO 0** non può essere utilizzata per saltare ad un gestore di errori, perché nell'ambito della gestione degli errori essa ha un significato particolare. **ON ERROR GOTO 0** produce infatti due effetti, a seconda del punto in cui appare. Se si trova all'esterno di una routine di gestione degli errori, disattiva la gestione degli errori. Se si trova invece all'interno della routine (come avviene nella routine `Gestore` nell'esempio seguente), il BASIC visualizza il messaggio standard relativo all'errore verificatosi ed il programma si arresta.

Di conseguenza, anche se il programma contiene una riga con numero 0, l'istruzione **ON ERROR GOTO 0** indica al BASIC di disattivare il rilevamento degli errori o di terminare l'esecuzione.

6.1.2 Scrittura di una routine per la gestione degli errori

Una routine per la gestione degli errori è composta di tre parti:

- L'etichetta o il numero di riga specificato dall'istruzione **ON ERROR GOTO** *riga*, che indica l'istruzione alla quale il programma salterà dopo un errore.
- La parte principale della routine, che analizza l'errore che ha causato il salto ed agisce in modo appropriato su ciascun errore previsto.
- Almeno una istruzione **RESUME** o **RESUME NEXT**, per riprendere il controllo dopo la routine.

Un gestore di errori va collocato in un punto in cui non può essere eseguito nell'ordine normale di esecuzione del programma. Ad esempio, una routine di gestione degli errori nel modulo principale si può mettere dopo un'istruzione **END**. Altrimenti, potrebbe essere necessaria un'istruzione **GOTO** per saltarla durante la normale esecuzione.

6.1.2.1 Utilizzo di ERR per l'identificazione degli errori

Dopo che il programma è saltato ad una routine di gestione degli errori, esso deve stabilire l'errore che ha causato il passaggio. A questo scopo viene utilizzata la funzione **ERR**. Essa restituisce il codice numerico relativo all'errore più recente verificatosi nel corso dell'esecuzione. (La tabella I.1, "Codici degli errori nel corso di esecuzione", contiene un elenco completo dei codici relativi agli errori nel corso dell'esecuzione.)

Nota Gli errori che accadono all'interno di una routine per la gestione degli errori non possono essere gestiti. Se si verifica un errore mentre una routine ne sta elaborando un altro, il programma visualizza il messaggio relativo al nuovo errore e si interrompe. Inoltre, durante l'esecuzione di un gestore di errori viene sospesa la gestione degli eventi, che viene ripristinata quando QuickBASIC ritorna dal gestore di errori. Il primo evento verificatosi dopo la sospensione della gestione degli eventi viene memorizzato.

Esempio

Il programma seguente include la routine di gestione degli errori `Gestore`, che gestisce tre diverse situazioni di errore. La funzione **ERR** utilizzata in un'istruzione **SELECT CASE** permette alla routine di agire in maniera appropriata a ciascun errore.

```
DATA BASIC, Pascal, FORTRAN, C, Modula, Forth
DATA LISP, Ada, COBOL

CONST FALSO = 0, VERO = NOT FALSO

FineDeiDati = FALSO       ' Imposta il flag fine dati.
```

```
ON ERROR GOTO Gestore      ' Attiva la gestione degli errori.

' Predispone la stampante per l'output:
OPEN "LPT1:" FOR OUTPUT AS #1

DO
   ' Continua a leggere gli elementi dalle istruzioni DATA
   ' precedenti, e a stamparli sulla stampante, finché non
   ' compare il messaggio di errore "Dati esauriti":
   READ Buffer$
   IF NOT FineDeiDati THEN
      PRINT #1, Buffer$
   ELSE
      EXIT DO
   END IF
LOOP

CLOSE #1
END

Gestore:            ' Routine di gestione degli errori.

' Utilizza ERR per determinare quale errore ha causato
' il passaggio a "Gestore":
SELECT CASE ERR
   CASE 4

      ' 4 è il codice del messaggio "Dati esauriti" nelle
      ' istruzioni DATA:
      FineDeiDati = VERO
      RESUME NEXT
   CASE 25

      ' 25 è il codice dell' errore "Difetto di periferica",
      ' che può essere causato nel tentativo di restituire
      ' un output alla stampante quando non è accesa:
      PRINT "Accendere la stampante, quindi ";
      PRINT "premere un tasto per continuare"
      Pausa$ = INPUT$(1)
      RESUME
   CASE 27

      ' 27 è il codice per "Carta esaurita":
      PRINT "E' finita la carta nella stampante. Inserire ";
      PRINT "la carta, e premere un tasto per continuare."
      Pausa$ = INPUT$(1)
```

```
      ' Inizia la lettura dei dati dall'inizio della prima
      ' istruzione DATA dopo l'inserimento della carta:
      RESTORE
      RESUME
   CASE ELSE

      ' Si è verificato un errore imprevisto; visualizza
      ' il relativo messaggio di errore ed interrompe
      ' l'esecuzione del programma:
      ON ERROR GOTO 0
   END SELECT
```

6.1.2.2 Ritorno da una routine per la gestione degli errori

L'istruzione **RESUME** riprende il controllo da una routine di gestione degli errori. Nell'esempio precedente sono state utilizzate le seguenti due forme di **RESUME** per ritornare da `Gestore`:

Istruzione	*Azione*
RESUME	Determina il ritorno del programma all'istruzione che ha provocato l'errore. Se il programma ha dovuto cercare un gestore di errori attivo in un altro modulo, il controllo ritorna all'ultima istruzione eseguita in quel modulo. Il programma precedente ha utilizzato **RESUME** per ritornare all'istruzione **PRINT** che tentava di inviare l'output alla stampante, offrendo al programma un'altra possibilità di stampare il valore di `Buffer$` dopo la presumibile accensione della stampante.
RESUME NEXT	Determina il ritorno del programma all'istruzione successiva a quella che ha causato l'errore. Se il programma ha dovuto cercare un gestore di errori attivo in un altro modulo, il controllo ritorna all'istruzione successiva all'ultima eseguita in quel modulo. Il programma precedente ha utilizzato l'istruzione **RESUME NEXT** durante la correzione dell'errore `Dati esauriti`. In questo caso, una semplice **RESUME** avrebbe introdotto il programma in un ciclo senza fine, perché ogni volta che il controllo fosse tornato all'istruzione **READ** del programma principale, sarebbe risultato un altro errore `Dati esauriti`, che avrebbe richiamato ripetutamente la routine.

Nell'illustrazione 6.1 si vede l'ordine di esecuzione del programma durante la gestione degli errori con **RESUME**.

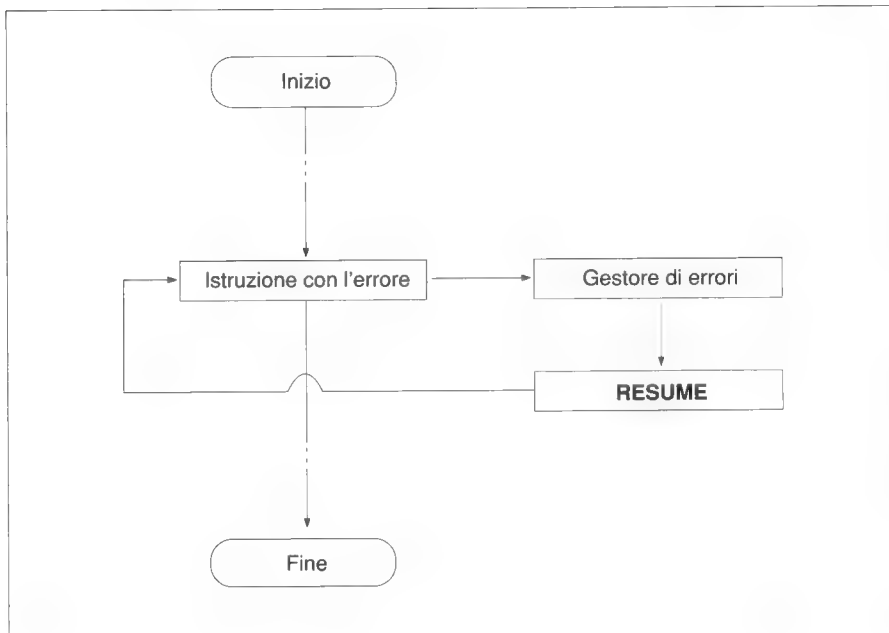
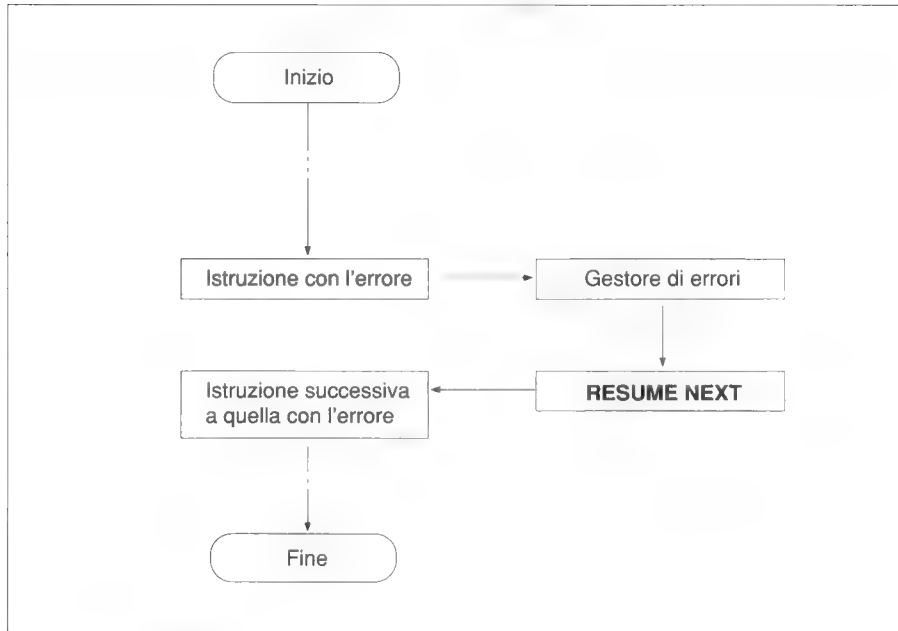
Illustrazione 6.1 Ordine di esecuzione del programma con RESUME

Confrontare la precedente illustrazione con l'illustrazione 6.2, dove si vede cosa accade in un programma che utilizza **RESUME NEXT**.

Illustrazione 6.2 Ordine di esecuzione del programma con RESUME NEXT

Un'ulteriore variante di **RESUME** è **RESUME** *riga*, dove *riga* è un numero o un'etichetta al di fuori di qualunque blocco **SUB...END SUB**, **FUNCTION...END FUNCTION**, o **DEF FN...END DEF**. Poiché è necessario che *riga* si trovi al di fuori di tali blocchi, un'istruzione **RESUME** *riga* può produrre effetti inaspettati, se essa appare in una routine per la gestione degli errori dentro una procedura **SUB** o **FUNCTION** o in una funzione **DEF FN**. E' comunque preferibile l'uso di **RESUME** o **RESUME NEXT**.

6.2 Gestione degli eventi

La sezione 6.2.1 confronta due metodi utilizzabili dai programmi BASIC per rilevare gli eventi: la scansione e la gestione. Le sezioni 6.2.2–6.2.4 trattano i diversi eventi gestibili dal BASIC, e il modo in cui impostare e attivare una gestione, sospenderla o disattivarla. Le sezioni 6.2.5–6.2.7 forniscono informazioni più dettagliate relative alla gestione della pressione dei tasti e degli eventi di suono.

6.2.1 Rilevamento di eventi per mezzo di scansione

Con la "scansione" si può rilevare un evento e passare il controllo del programma alla routine appropriata. Durante la scansione, il programma interrompe qualsiasi operazione stia eseguendo al momento e aspetta esplicitamente un evento. Ad esempio, il ciclo seguente continua a controllare la tastiera finché l'utente non preme i tasti U o N:

```
DO
   Prova$ = UCASE$(INKEY$)
LOOP UNTIL Prova$ = "N" OR Prova$ = "U"
```

La scansione è utile quando si conosce già il punto esatto nell'ordine di esecuzione del programma in cui si desidera rilevare un evento.

6.2.2 Rilevamento di eventi per mezzo di gestori

Se non si vuole che il programma faccia una pausa, oppure si desidera che esso effettui un controllo continuo tra un'istruzione e l'altra, la scansione risulta lenta, se non addirittura impossibile, e la gestione diventa l'alternativa migliore.

Esempio

L'esempio seguente mostra come utilizzare la gestione degli eventi:

```
' Avvisa il programma di passare alla "SubTasto"
' ogni volta che l'utente preme il tasto funzione F1:
ON KEY(1) GOSUB SubTasto

' Attiva la gestione del tasto F1. Il BASIC ora controlla
' questo evento nel sottofondo sino al termine del
' programma, o fino alla disattivazione della gestione
' per mezzo di KEY(1) OFF o alla sua sospensione con
' KEY(1) STOP:
KEY(1) ON
```

```
OPEN "DATI" FOR INPUT AS #1
OPEN "DATI.OUT" FOR OUTPUT AS #2

DO UNTIL EOF(1)
   LINE INPUT #1, BufferRiga$
   PRINT #2, BufferRiga$
LOOP

SubTasto:          ' Quando l'utente preme F1, il programma
.                  ' salta a questa procedura.
.
.
RETURN
```

6.2.3 Specificazione di eventi da rilevare e attivazione della gestione degli eventi

L'esempio precedente ha individuato tre fasi necessarie alla gestione degli eventi:

1. Definire una subroutine di destinazione, nota come "subroutine di gestione degli eventi".
2. Indicare al programma di saltare alla subroutine di gestione degli eventi al verificarsi di un evento, per mezzo di un'istruzione **ON** *evento* **GOSUB**.
3. Attivare la gestione dell'evento con un'istruzione *evento* **ON**.

Il programma non può gestire un dato *evento* finché non incontra sia un'istruzione *evento* **ON** che un'istruzione **ON** *evento* **GOSUB**.

Una routine di gestione degli eventi è una ordinaria procedura **GOSUB** BASIC: il controllo passa alla prima riga della routine ogni volta che l'evento viene rilevato e, per mezzo dell'istruzione **RETURN**, ritorna al livello principale del modulo in cui si trova la routine.

Notare le importanti differenze tra la sintassi della gestione degli errori e quella della gestione degli eventi:

Gestione degli errori	***Gestione degli eventi***
Viene attivata con l'istruzione **ON ERROR GOTO**, che specifica anche la prima riga della routine di gestione degli errori.	Viene attivata con l'istruzione *evento* **ON**; un'istruzione a parte, **ON** *evento* **GOSUB**, specifica la prima riga della subroutine di gestione degli eventi.
Riprende il controllo dalla routine degli errori per mezzo di una o più istruzioni **RESUME**, **RESUME NEXT**, o **RESUME** *riga*.	Riprende il controllo dalla subroutine degli eventi per mezzo di una o più istruzioni **RETURN**.

6.2.4 Eventi che il BASIC può gestire

All'interno di un programma BASIC si possono gestire i seguenti eventi:

Istruzione BASIC	*Evento gestito*
COM (*numeroporta*)	L'arrivo di dati da una delle porte seriali (1 o 2) al buffer di comunicazione (l'area di memoria temporanea per i dati inviati o ricevuti dalla porta seriale)
KEY (*numerotasto*)	La pressione di un determinato tasto
PEN	L'attivazione della penna ottica
PLAY (*limitecoda*)	Lo scendere al di sotto di *limitecoda* del numero di note rimaste da suonare nel sottofondo
STRIG (*numeropulsante*)	La pressione del pulsante di azione del joystick
TIMER (*intervallo*)	Lo scadere di *intervallo* secondi

6.2.5 Sospensione e disattivazione del rilevamento di errori

Il rilevamento e la gestione di qualunque evento si possono disattivare per mezzo dell'istruzione *evento* **OFF**. Gli eventi che si verificano dopo l'esecuzione di questa istruzione vengono ignorati. Se si vuole sospendere la gestione di un dato evento, ma continuarne il rilevamento, basta utilizzare l'istruzione *evento* **STOP**.

Se dopo questa istruzione si verifica un *evento*, non si attua il passaggio alla routine di gestione degli errori. Il programma però memorizza l'evento verificatosi, e, non appena la gestione viene riattivata con *evento* **ON**, si attua immediatamente il passaggio alla routine.

Ogni volta che il programma entra in una routine di gestione degli eventi, questa esegue un'implicita istruzione *evento* **STOP** relativa al dato evento; nel momento in cui il programma riprende il controllo dalla routine, esegue un'implicita istruzione *evento* **ON** per l'evento. Ad esempio, se una routine di gestione dei tasti sta elaborando la pressione di un tasto, la gestione dello stesso tasto viene sospesa fino al completamento dell'elaborazione della pressione precedente. Se nel frattempo il tasto viene nuovamente premuto, l'ulteriore pressione viene memorizzata, e gestita dopo che il controllo ritorna dalla routine di gestione dei tasti.

Esempio

Una gestione degli eventi interrompe qualsiasi operazione in corso all'interno del programma nel momento in cui si verifica l'evento. Di conseguenza, è necessario far precedere da un'istruzione *evento* **STOP** e seguire da un'istruzione *evento* **ON** quelle istruzioni che non si vogliono interrompere, come in questo esempio:

```
' Ogni minuto (60 secondi), saltare alla subroutine
' IndicaOra:
ON TIMER (60) GOSUB IndicaOra

' Attiva la gestione dello scadere dei 60 secondi:
TIMER ON
.
.
.
TIMER STOP          ' Sospende la gestione.
.                   ' Una sequenza di righe che non si
.                   ' vogliono interrompere, anche se
.                   ' sono trascorsi 60 o più secondi.
TIMER ON            ' Riattiva la gestione.
.
.
.
END

IndicaOra:

   ' Legge la posizione corrente del cursore,
   ' e la memorizza nelle variabili Riga e Colonna:
   Riga = CSRLIN
   Colonna = POS(0)

   ' Visualizza l'ora alla riga 24, colonna 20:
   LOCATE 24,20
   PRINT TIME$

   ' Riporta il cursore alla sua posizione d'origine
   ' e ritorna al programma principale:
   LOCATE Riga, Colonna
RETURN
```

6.2.6 Gestione della pressione dei tasti

Per rilevare una pressione di tasto e trasferire il controllo del programma ad una routine di pressione dei tasti, sono necessarie le seguenti istruzioni:

ON KEY (*numerotasto*) **GOSUB** *riga*
KEY (*numerotasto*) **ON**

Qui il valore di *numerotasto* corrisponde ai seguenti tasti:

Valore	*Tasto*
1-10	Tasti funzione F1-F10 (talvolta detti "tasti programmati")
11	Il tasto di direzione SU
12	Il tasto di direzione SINISTRA
13	Il tasto di direzione DESTRA
14	Il tasto di direzione GIU'
15-25	Tasti definiti dall'utente (consultare le sezioni 6.2.6.1-6.2.6.2)
30	Il tasto funzione F11 (tastiere avanzate)
31	Il tasto funzione F12 (tastiere avanzate)

Esempi

Le due righe seguenti fanno saltare il programma alla routine `SubTasto` ogni volta che viene premuto il tasto funzione F2:

```
ON KEY(2) GOSUB SubTasto
KEY(2) ON
.
.
.
```

Le quattro righe seguenti fanno saltare il programma alla routine `TastoGiu` quando viene premuto il tasto di direzione GIU', e a `TastoSu` quando si preme il tasto di direzione SU:

```
ON KEY(11) GOSUB TastoSu
ON KEY(14) GOSUB TastoGiu
KEY(11) ON
KEY(14) ON
.
.
.
```

6.2.6.1 Gestione dei tasti definiti dall'utente

Oltre a fornire i numeri preassegnati dei tasti da 1 a 14 (più il 30 e il 31 nelle tastiere avanzate), il BASIC consente di assegnare i numeri da 15 a 25 a qualunque dei rimanenti tasti. Per definire un tasto da gestire, utilizzare queste tre istruzioni:

KEY *numerotasto*, **CHR$ (0) + CHR$** (*codicescansione*)
ON KEY (*numerotasto*) **GOSUB** *riga*
KEY (*numerotasto*) **ON**

Qui *numerotasto* rappresenta un valore da 15 a 25, e *codicescansione* è il codice di scansione del tasto. (I codici sono riportati nella prima colonna della tabella "Codici della tastiera", nell'appendice D.) Ad esempio, le righe seguenti fanno saltare il programma alla routine `TastoT` ogni volta che l'utente preme il tasto T:

```
' Definisce il tasto 15 (il codice di scansione di "t" è il
' 20 decimale):
KEY 15, CHR$(0) + CHR$(20)

' Definisce il gestore (dove saltare quando viene premuta
' la "t"):
ON KEY(15) GOSUB TastoT

KEY(15) ON                           ' Attiva il rilevamento
                                     ' del tasto 15.

PRINT "Premere u per uscire."

DO                                   ' Ciclo inattivo: attende
LOOP UNTIL INKEY$ = "u"              ' che l'utente prema "u".

END

TastoT:                              ' Subroutine gestione
   PRINT "E' stata premuta la t."    ' tasti
RETURN
```

6.2.6.2 Gestione di combinazioni di tasti definiti dall'utente

E' possibile la gestione anche dei tasti "combinati". Un tasto si dice "combinato" quando viene premuto contemporaneamente ad uno o più dei tasti speciali MAIUSC, CTRL, o ALT, oppure dopo aver attivato i tasti BLOC NUM o BLOC MAIUS.

Ecco come gestire le combinazioni di tasti:

KEY *numerotasto*, **CHR$** (*flagtastiera*) + **CHR$** (*codicescansione*)
ON KEY (*numerotasto*) **GOSUB** *riga*
KEY (*numerotasto*) **ON**

Qui *numerotasto* rappresenta un valore da 15 a 25; *codicescansione* è il codice di scansione del tasto primario; *flagtastiera* è la somma dei singoli codici dei tasti speciali premuti, come mostra l'elenco successivo:

Tasto	*Codice di flagtastiera*
MAIUSC	1, 2, o 3 La gestione dei tasti considera uguali i tasti MAIUSC sinistro e destro, perciò si può gestire il tasto MAIUSC con 1 (sinistro), 2 (destro), o 3 (sinistro + destro).
CTRL	4
ALT	8
BLOC NUM	32
BLOC MAIUS	64
Qualunque tasto avanzato della tastiera avanzata (quali SINISTRA O CANC non sul tastierino numerico)	128

Ad esempio, le istruzioni seguenti attivano la gestione di CTRL+S. Notare che queste istruzioni hanno lo scopo di gestire sia la combinazione CTRL+S (minuscola) che CTRL+MAIUSC+S (maiuscola). Per gestire la S maiuscola, è necessario che il programma riconosca le maiuscole generate sia dalla pressione del tasto MAIUSC che dall'attivazione del tasto BLOC MAIUS, come si vede qui:

```
' 31 è il codice di scansione del tasto S;
' 4 è il codice del tasto CTRL:
KEY 15, CHR$(4) + CHR$(31)      ' Gestisce CTRL+S.

' 5 è il codice del tasto CTRL + il codice del tasto MAIUSC:
KEY 16, CHR$(5) + CHR$(31)      ' Gestisce CTRL+MAIUSC+S.

' 68 è il codice del tasto CTRL + il codice del
' tasto BLOCMAIUS:
KEY 17, CHR$(68) + CHR$(31)     ' Gestisce CTRL+BLOCMAIUS+S.
```

```
' Indica al programma dove saltare (nota: stessa subroutine
' per ciascun tasto):
ON KEY (15) GOSUB GestisceCtrl
ON KEY (16) GOSUB GestisceCtrl
ON KEY (17) GOSUB GestisceCtrl

Attiva il rilevamento per le tre combinazioni tasti:
KEY (15) ON
KEY (16) ON
KEY (17) ON
.
.
.
```

Le istruzioni seguenti attivano la gestione di CTRL+ALT+CANC:

```
' 12 = 4 + 8 = (codice tasto CTRL) + (codice tasto ALT)
' 83 = codice di scansione del tasto CANC
KEY 20, CHR$(12) + CHR$(83)
ON KEY(20) GOSUB GestoreTasti
KEY(2) ON
.
.
.
```

In quest'esempio la gestione degli eventi BASIC preclude l'effetto normale di CTRL+ALT+CANC (il riavvio del sistema). Utilizzando questo gestore nel programma, si evita il riavviamento accidentale del sistema da parte dell'utente durante l'esecuzione del programma.

Utilizzando una tastiera avanzata, è possibile gestire qualunque tasto sul tastierino riservato, assegnando la stringa

CHR$(128) + CHR$(*codicescansione*)

a qualsiasi *numerotasto* da 15 a 25.

L'esempio successivo spiega come gestire i tasti di direzione SINISTRA sia sul tastierino riservato del cursore che su quello numerico.

```
' 128 = flag dei tasti del tastierino riservato;
' 75 = codice del tasto freccia SINISTRA:
KEY 15, CHR$(128) + CHR$(75)            ' Gestisce il tasto
ON KEY(15) GOSUB TastierinoCursore      ' SINISTRA sul
KEY(15) ON                              ' tastierino riservato
                                        ' del cursore.
ON KEY(12) GOSUB TastierinoNumerico     ' Gestisce il tasto
KEY(12) ON                              ' SINISTRA sul
                                        ' tastierino numerico.
DO: LOOP UNTIL INKEY$ = "u"             ' Avvia un ciclo
                                        ' inattivo.
END

TastierinoCursore:
   PRINT " E' stato premuto il tasto SINISTRA sul";
   PRINT " tastierino del cursore."
RETURN

TastierinoNumerico:
   PRINT " E' stato premuto il tasto SINISTRA sul";
   PRINT " tastierino numerico."
RETURN
```

Importante Per compatibilità, QuickBASIC adotta molte convenzioni del BASICA. Una di queste riguarda il modo in cui vengono gestiti i tasti FUNZIONE.

Se la gestione di un tasto FUNZIONE è stata attivata per mezzo di un'istruzione **ON KEY** (*n*) **GOSUB**, sia il BASICA che QuickBASIC rilevano F*n* indipendentemente dal fatto che sia stato o meno combinato (con CTRL, ALT, o MAIUSC). Ciò significa che vengono ignorati eventuali gestori definiti dall'utente di tasti FUNZIONE combinati.

Di conseguenza, per gestire separatamente il tasto FUNZIONE da solo e quello combinato, creare definizioni utente per entrambi i casi. I tasti che non sono mai utilizzati in combinazione si possono gestire con l'istruzione **ON KEY** (*n*) **GOSUB**.

6.2.7 Gestione degli eventi di suono

Con l'istruzione per la generazione di suoni **PLAY** si può far eseguire la musica in primo piano o nel sottofondo. Se si sceglie l'opzione primo piano (la predefinita), non può verificarsi nient'altro finché il brano non è terminato. Utilizzando invece l'opzione **MB** nella stringa di suono **PLAY**, il motivo suona nel sottofondo, mentre continua l'esecuzione delle successive istruzioni del programma.

Per eseguire un brano musicale nel sottofondo, l'istruzione **PLAY** memorizza in un buffer fino a 32 note per volta, e le suona mentre il programma esegue altre operazioni. Una "gestione del suono" conta il numero di note rimaste nel buffer: non appena questo scende sotto il limite impostato, il programma salta alla prima riga della routine di gestione del suono specificata.

Per impostare nel programma una gestione di suono, sono necessarie le seguenti istruzioni:

ON PLAY (*limite***) GOSUB** *riga*
PLAY ON
PLAY "MB"
.
.
.
PLAY *stringamusica*
[PLAY *stringamusica***]**
.
.
.

Qui *limite* rappresenta un numero tra 1 e 32. Ad esempio, questo brano fa saltare il programma alla subroutine `GestioneSuono` ogni volta che il numero delle note nel buffer di suono passa da 8 a 7:

```
ON PLAY(8) GOSUB GestioneSuono
PLAY ON
.
.
.
PLAY "MB"    ' Suona le rimanenti note nel sottofondo.
PLAY "o1 A# B# C-"
.
.
.
GestioneSuono:
   ' Subroutine gestione suono
   .
   .
   .
RETURN
```

Importante Un gestore di suono viene attivato solo quando il numero di note va da limite a limite −1. Ad esempio, se il buffer di suono nell'esempio precedente non contiene mai più di 7 note, la gestione non avrà mai luogo. Nell'esempio, essa ha luogo soltanto quando il numero di note va da 8 a 7.

Una subroutine di gestione del suono può essere utilizzata per eseguire ripetutamente lo stesso brano musicale durante l'esecuzione del programma, come è mostrato nell'esempio seguente:

```
' Attiva la gestione degli eventi suono sottofondo:
PLAY ON

' Passa alla subroutine Ripristina quando ci sono meno
' di due note nel buffer di suono:
ON PLAY(2) GOSUB Ripristina

PRINT "Premere un tasto per iniziare, o u per uscire."
Pausa$ = INPUT$(1)

' Seleziona l'opzione sottofondo di PLAY:
PLAY "MB"

' Inizia a suonare: le note vengono collocate nel buffer
' per la musica nel sottofondo:
GOSUB Ripristina

I = 0

DO

   ' Continua a visualizzare i numeri da 0 a 10000
   ' finché l'utente non preme il tasto "u". Nel
   ' frattempo, la musica continua nel sottofondo:
   PRINT I
   I = (I + 1) MOD 10001
LOOP UNTIL INKEY$ = "u"

END

Ripristina:
   ' Suona il motivo iniziale della Quinta di Beethoven:
   Ascolta$ = "t180 o2 p2 p8 L8 GGG L2 E-"
   Destino$ = "p24 p8 L8 FFF L2 D"
   PLAY Ascolta$ + Destino$
RETURN
```

6.3 Gestione di errori ed eventi nelle procedure SUB o FUNCTION

Nell'utilizzo di gestori di errori o di eventi nelle procedure BASIC, ricordare che queste due istruzioni possono apparire all'interno di un blocco **SUB...END SUB** o **FUNCTION...END FUNCTION**:

ON ERROR GOTO *riga*
ON *evento* **GOSUB** *riga*

Tuttavia, *riga* deve identificare una riga del codice a livello di modulo, e non all'interno della **SUB** o **FUNCTION**.

Esempio

L'esempio successivo indica il punto dove inserire una routine di gestione degli errori che elabori errori rilevati all'interno di una procedura **SUB**:

```
CALL SubBreve
END

CatturaErrori:
   ' Inserire la routine CatturaErrori a livello del modulo,
   ' al di fuori del sottoprogramma:
   PRINT "Errore" ERR "rilevato dal gestore."
RESUME NEXT

SUB SubBreve STATIC
   ON ERROR GOTO CatturaErrori
   ERROR 62
END SUB
```

Output

```
Errore 62 rilevato dal gestore.
```

6.4 Gestione tra moduli multipli

Nelle versioni QuickBASIC precedenti alla 4.5, tra un modulo e l'altro potevano essere gestiti soltanto gli eventi. Una volta definita una routine di gestione degli eventi ed attivata la gestione di un evento in un modulo, il verificarsi di quell'evento in qualsiasi altro modulo, durante l'esecuzione del programma, attivava un salto alla routine.

Gli errori invece non potevano essere gestiti tra moduli. Se si verificava un errore in un modulo sprovvisto di un gestore di errori attivo, l'esecuzione del programma si interrompeva, anche se in un altro modulo esisteva un gestore in grado di occuparsene.

La versione 4.5 QuickBASIC allarga l'area di validità dei gestori degli errori: tra moduli diversi possono essere gestiti sia eventi che errori. Le due sezioni successive spiegano il funzionamento e chiariscono le rimanenti differenze tra la gestione degli eventi e quella degli errori.

6.4.1 Gestione degli eventi tra moduli

L'output del programma seguente dimostra che una gestione impostata per il tasto funzione F1 nel modulo principale rimane attiva anche quando il programma sta eseguendo un altro modulo:

```
'===========================================================
'                          MODULO
' ==========================================================

ON KEY(1) GOSUB TastoF1
KEY (1) ON
PRINT "Nel modulo principale. Premere c per continuare."
DO: LOOP UNTIL INKEY$ = "c"

CALL SubTasto

PRINT "Ritornato al modulo principale. Premere t per ";
PRINT "terminare."
DO: LOOP UNTIL INKEY$ "t"
END

TastoF1:
   PRINT "Pressione del tasto F1 gestita dal modulo ";
   PRINT "principale."
RETURN
```

```
' ===========================================================
'                      MODULO SUBTASTO
' ===========================================================

SUB SubTasto STATIC
   PRINT "Nel modulo contenente SUBTASTO. Premere r per ";
   PRINT "ritornare."

   ' Premendo F1 da qui richiama sempre la subroutine
   ' TastoF1 nel modulo PRINCIPALE:
   DO: LOOP UNTIL INKEY$ = "r"
END SUB
```

Output

```
Nel modulo principale. Premere c per continuare.
Pressione del tasto F1 gestita dal modulo principale.
Nel modulo contenente SUBTASTO. Premere r per ritornare.
Ritornato al modulo principale. Premere t per terminare.
Pressione del tasto F1 gestita dal modulo principale.
```

6.4.2 Gestione degli errori tra moduli

Gli errori possono essere gestiti tra più moduli. Se nel modulo in cui si è verificato l'errore non esiste alcun gestore di errori attivo, il BASIC inizia a cercarne uno nel modulo da cui è stato chiamato il modulo attivo, e poi continua a ritroso per la sequenza dei moduli chiamati, finché non ne trova uno in cui sia presente un gestore di errori attivo. Se non ne trova, esso visualizza un messaggio di errore e l'esecuzione del programma si arresta.

Notare che il BASIC non esegue la ricerca in tutti i moduli, ma solo in quelli lungo il percorso dei richiami eseguiti che conduce al codice che ha generato l'errore.

Perché BASIC cerchi un gestore, bisogna che venga eseguita un'istruzione **ON ERROR** prima che si verifichi l'errore. Di conseguenza, l'istruzione **ON ERROR GOTO** va inserita in un punto in cui può essere raggiunta dall'ordine di esecuzione del programma; ad esempio, all'interno di una procedura chiamata dal modulo principale (consultare gli esempi che seguono).

Esempi

L'esempio seguente mostra come gestire errori in una procedura di una libreria Quick. La procedura `TipoScheda` di questa libreria prova tutte le possibili istruzioni **SCREEN** per vedere quali modalità schermo grafiche supporti l'hardware del computer. (Per ulteriori informazioni relative alle librerie Quick, consultare l'appendice H, "Creazione ed utilizzo delle librerie Quick".)

```
' ======= CODICE A LIVELLO DI MODULO LIBRERIA QUICK ========
'  Le routine di gestione degli errori per le procedure
'  di questa libreria vanno definite a questo livello,
'  che viene eseguito solo quando si verifica un errore
'  in una procedura.
' ==========================================================

DEFINT A-Z

CONST FALSO = 0, VERO = NOT FALSO

.
.
.
CALL TipoScheda
.
.
.
END

ErroreSi:                           ' Routine gestione errori
   VisualizzaErrore = VERO
   RESUME NEXT

' ====== CODICE A LIVELLO DI PROCEDURA LIBRERIA QUICK ======
'  Gestione errori attivata a questo livello.
' ==========================================================

SUB TipoScheda STATIC
   ' La variabile VisualizzaErrore è condivisa con il
   ' gestore di errori ErroreSi listato nel codice sopra:
   SHARED VisualizzaErrore
   ' Dimensiona la matrice VisualizzaTipo:
   DIM VisualizzaTipo (1 TO 13)
```

```
    ' Inizializza il contatore degli indici della matrice
    ' VisualizzaTipo:
    J = 1

    ' Imposta la gestione degli errori:
    ON ERROR GOTO ErroreSi

    FOR Prova = 13 TO 1 STEP -1
       SCREEN Prova                     ' Prova se un'istruzione
                                        ' SCREEN causa errori.
       IF NOT VisualizzaErrore THEN     ' Nessun errore, è
                                        ' una valida modalità
                                        ' schermo.
       MostraTipo(J) = Prova            ' Memorizza la modalità
                                        ' nella matrice.
       J = J + 1                        ' Incrementa l'indice.
       ELSE
          VisualizzaErrore = FALSO      ' Errore; reimposta
                                        ' VisualizzaErrore.
       END IF
    NEXT Prova

    SCREEN 0, 0                         ' Imposta modalità testo
    WIDTH 80                            ' 80 colonne.

    LOCATE 5, 10
    PRINT " Il computer supporta queste modalità schermo:"
    PRINT
    FOR I = 1 TO J                      ' Visualizza le modalità
                                        ' che non generano
                                        ' errori.
       PRINT TAB(20); "SCREEN"; TipoScheda(I)
    NEXT I

    LOCATE 20, 10
    PRINT "Premere un tasto per continuare..."

    DO: LOOP WHILE INKEY$ = ""

END SUB
```

Se si vuole che la routine di gestione degli errori in ogni modulo agisca esattamente nello stesso modo, si può far chiamare a ciascuna routine la medesima procedura **SUB** per l'elaborazione degli errori, come illustrato nell'esempio che segue:

```
' ==========================================================
'                  MODULO PRINCIPALE
' ==========================================================

DECLARE SUB GestGlobale(NomeModulo$)
DECLARE SUB SubBreve()

ON ERROR GOTO GestLocale
ERROR 57                ' Simula il verificarsi dell'errore 57
                        ' (" Errore di I/O su periferica").
SubBreve                ' Richiama la SUB SubBreve.
END

GestLocale:
   NomeModulo$ = "PRINCIPALE"
   CALL GestGlobale(NomeModulo$)    ' Richiama la
                                    ' SUB GestGlobale.
RESUME NEXT

' ==========================================================
'                     MODULO SUBBREVE
' ==========================================================

DECLARE SUB GestGlobale(NomeModulo$)

GestLocale:
   NomeModulo$ = "SUBBREVE"
   GestGlobale NomeModulo$     ' Richiama la GestGlobale.
RESUME NEXT

SUB SubBreve STATIC
   ON ERROR GOTO GestLocale
   ERROR 13                    ' Simula l'errore "Mancata
                               ' corrispondenza dei tipi".
END SUB
```

```
' ============================================================
                     MODULO GESTGLOBALE
' ============================================================
```

```
SUB GestGlobale(NomeModulo$) STATIC
   PRINT "Errore";ERR;"gestito nel modulo";NomeModulo$
END SUB
```

Output

```
Errore 57 gestito nel modulo PRINCIPALE
Errore 13 gestito nel modulo SUBBREVE
```

6.5 Gestione di errori ed eventi in programmi compilati con il comando BC

Se sono vere entrambe le condizioni seguenti, la compilazione di un programma richiede le opzioni della riga di comando BC indicate qui sotto:

- Il programma è stato steso al di fuori dell'ambiente QuickBASIC, cioè utilizzando un altro editor di testo per digitare il codice sorgente BASIC e, da questo codice, si sta creando un programma eseguibile autonomo per mezzo dei comandi BC e LINK.
- Il programma contiene una delle istruzioni elencate nella seconda colonna della seguente tabella.

Tabella 6.1 Opzioni della riga di comando BC per la gestione di errori ed eventi

Opzione riga di comando	*Istruzioni*	*Spiegazione*
/E	**ON ERROR GOTO** **RESUME** *riga*	L'opzione /E indica al compilatore che il programma stesso gestisce gli errori con le istruzioni **ON ERROR GOTO** e **RESUME**.
/V	**ON** *evento* **GOSUB** *evento* **ON**	L'opzione /V indica al programma di controllare tra un'istruzione e l'altra se ha avuto luogo l'evento dato (confrontare con l'effetto di /W).

continua

Opzione riga di comando	*Istruzioni*	*Spiegazione*
/W	**ON** *evento* **GOSUB** *evento* **ON**	L'opzione /W indica al programma di controllare tra una riga e l'altra se ha avuto luogo l'evento dato (confrontare con l'effetto di /V). Poiché il BASIC permette più istruzioni su una sola riga, i programmi compilati con l'opzione /W potrebbero effettuare un controllo meno frequente di quelli compilati con l'opzione /V.
/X	**RESUME** **RESUME NEXT** **RESUME 0**	L'opzione /X indica al compilatore che nel programma è stata utilizzata una di queste forme di **RESUME** per riprendere il controllo da una routine per la gestione degli errori.

Esempio

Le seguenti righe di comando del DOS compilano e sottopongono al linker un modulo BASIC chiamato TOGLTAB.BAS, creando il programma autonomo TOGLTAB.EXE. (Dato che il programma viene compilato senza l'opzione /O, la sua esecuzione richiede la libreria di esecuzione BRUN45.LIB. Per ulteriori informazioni relative alla compilazione ed il linkaggio, consultare l'appendice G, "Compilazione ed esecuzione del link da DOS".) Poiché questo modulo contiene le istruzioni **ON ERROR GOTO** e **RESUME NEXT**, per la compilazione sono richieste le opzioni /E e /X.

```
BC TOGLTAB , , /E /X;
LINK TOGLTAB;
```

6.6 Esempio di applicativo: gestione di errori di accesso ai file (FILERR.BAS)

L'input del programma seguente è il nome di un file e una stringa da ricercare nel file. Esso elenca, quindi, tutte le righe del file contenenti la stringa specificata. Se si verifica un errore di accesso al file, questo viene gestito nella routine `ProcError`.

Istruzioni e funzioni utilizzate

Questo programma mostra l'utilizzo delle seguenti istruzioni e funzioni di gestione degli errori:

- **ERR**
- **ON ERROR GOTO**
- **RESUME**
- **RESUME NEXT**

Listato del programma

```
' Dichiara le costanti simboliche usate nel programma:
CONST FALSO = 0, VERO = NOT FALSO

DECLARE FUNCTION LetturaNomeFile$()

' Imposta la gestione di errori, e specifica il nome della
' routine gestione errori:
ON ERROR GOTO GestErr

DO
   Ricomincia = FALSO
   CLS

   NomeFile$ = LetturaNomeFile$      ' Legge il nome del file.

   IF NomeFile$ = "" THEN
      END                            ' Termina se viene
   ELSE                              ' premuto <INVIO>.
```

```
      ' Altrimenti apre il file, assegnandogli il primo
      ' numero di file disponibile:
      NumFile = FREEFILE
      OPEN NomeFile$ FOR INPUT AS NumFile
   END IF

   IF NOT Ricomincia THEN

      ' Legge la stringa da cercare:
      LINE INPUT "Digitare la stringa da cercare: _
         ", StringaCerca$
      StringaCerca$ = UCASE$(StringaCerca$)

      ' Cicla per le righe del file, stampandole
      ' se contengono la stringa da cercare:
      NumRiga = 1
      DO WHILE NOT EOF(NumFile)

         ' Legge una riga dal file:
         LINE INPUT #NumFile, BufferRiga$

         ' Controlla se c'è la stringa; se c'è stampa la
         ' riga e il relativo numero:
         IF INSTR(UCASE$(BufferRiga$), StringaCerca$) <> _
            0 THEN
            PRINT USING "#### &"; NumRiga; BufferRiga$
         END IF

         NumRiga = NumRiga + 1
      LOOP

      CLOSE NumFile                    ' Chiude il file.

   END IF
LOOP WHILE Ricomincia = VERO

END

GestErr:

   SELECT CASE ERR
```

```
    CASE 64:                          ' Nome di file non valido
       PRINT "** ERRORE - Nome di file non valido"

       ' Ottiene un nuovo nome di file e riprova:
       NomeFile$ = LetturaNomeFile$

       ' Riprende dall'istruzione che aveva causato
       ' l'errore:
       RESUME

    CASE 71:                          ' Disco non pronto
       PRINT "** ERRORE - Unità disco non pronta"
       PRINT "Premere c per continuare, r per ";
       PRINT "ricominciare, u per uscire: "
       DO
          Car$ = UCASE$(INPUT$(1))
          IF Car$ = "C" THEN
             RESUME                   ' Riprende da dove aveva
                                      ' interrotto

          ELSEIF Car$ = "R" THEN
             Ricomincia = VERO        ' Ricomincia dall'inizio
             RESUME NEXT

          ELSEIF Car$ = "U" THEN
             END                      ' Termina l'esecuzione
          END IF
       LOOP

    CASE 53, 76:                      ' File o percorso non
                                      ' trovato
       PRINT "** ERRORE - File o percorso non trovato"
       NomeFile$ = LetturaNomeFile$
       RESUME

    CASE ELSE:                        ' Errore imprevedibile

       ' Disattiva la gestione di errori e visualizza il
       ' messaggio standard del sistema:
       ON ERROR GOTO 0
 END SELECT
```

```
' ==================== LetturaNomeFile$ ====================
'  Restituisce il nome di un file digitato dall'utente.
' ==========================================================

FUNCTION LetturaNomeFile$ STATIC
   INPUT "Cercare in quale file? (premere <INVIO> per _
      terminare): ", FTemp$
   LetturaNomeFile$ = FTemp$
END FUNCTION
```

7 Programmazione a moduli

Il presente capitolo spiega come ottenere un controllo maggiore sui programmi mediante la loro scomposizione in "moduli". I moduli hanno una potente capacità organizzativa in quanto consentono di dividere un programma in parti logicamente, secondo la loro funzione, evitando così di concentrare tutto il codice in un unico file.

Il capitolo spiega come utilizzare questi moduli per:

- Scrivere e testare nuove procedure indipendentemente dal resto del programma
- Creare librerie per procedure **SUB** e **FUNCTION** che possano essere aggiunte a qualsiasi programma futuro
- Unire routine scritte in linguaggi differenti (quali il C o il MASM) con programmi BASIC

7.1 Perché utilizzare i moduli?

Un modulo è un file che contiene una parte eseguibile del programma. Un programma completo può essere composto di un solo modulo, o suddiviso tra due o più moduli.

Scomponendo un programma in moduli, istruzioni collegate logicamente tra loro vengono collocate in file a parte. Questo tipo di organizzazione velocizza e semplifica i processi di stesura, di verifica, e di messa a punto.

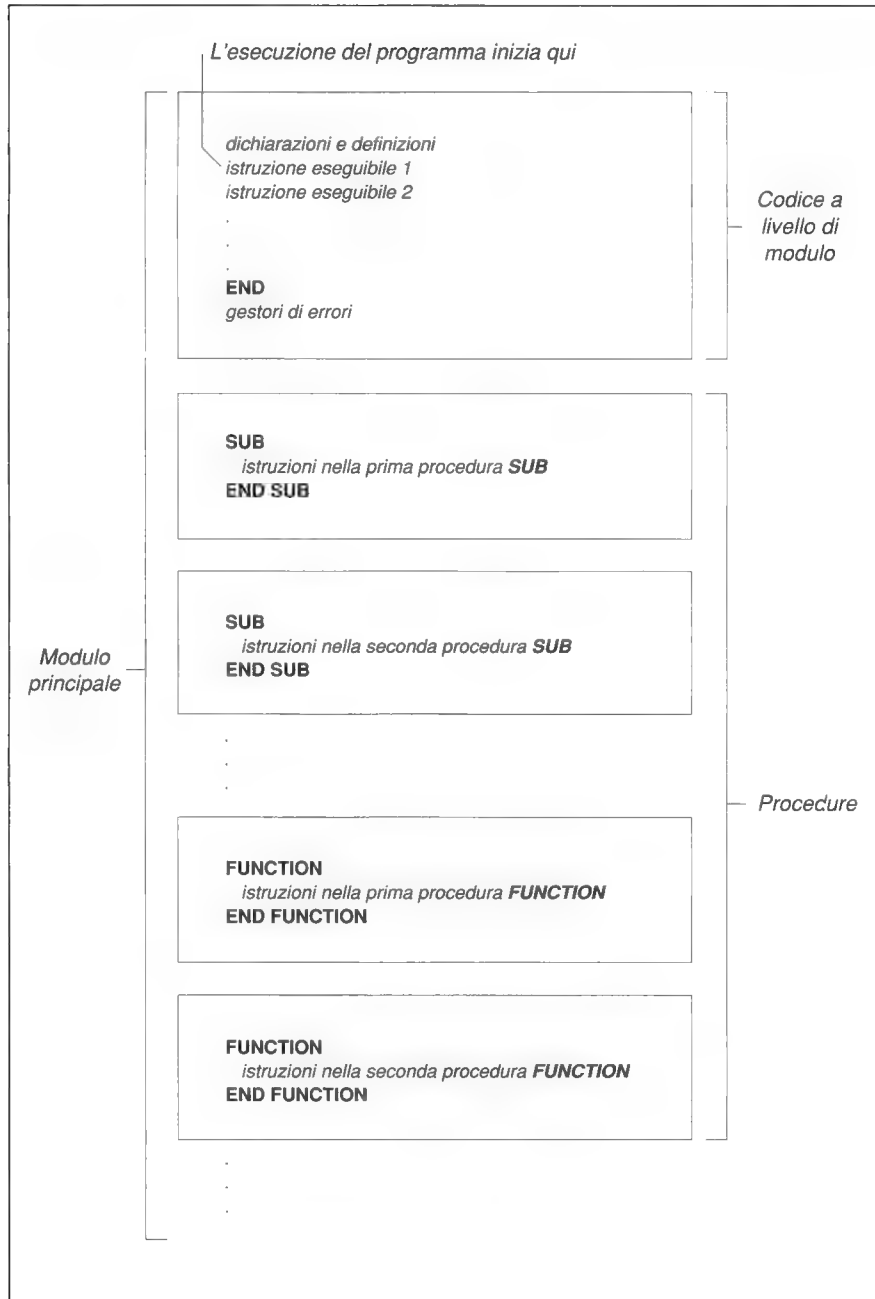
La scomposizione del programma in moduli offre questi vantaggi:

- I moduli consentono di scrivere le procedure indipendentemente dal resto del programma, e di unirle ad esso in un secondo momento. Questa disposizione è utile soprattutto perché le procedure si possono provare al di fuori dell'ambiente del programma.
- Due o più programmatori possono lavorare su diverse parti dello stesso programma senza interferire tra loro. Ciò è particolarmente utile nella gestione di programmi complessi.
- Quando si creano delle procedure che rispondono a personali esigenze di programmazione, esse possono essere riunite in un modulo, per essere poi riutilizzate in programmi nuovi semplicemente caricando quel modulo.
- Una programmazione a più moduli facilita la manutenzione del software. In un unico modulo di libreria può trovarsi una procedura utilizzata contemporaneamente da molti programmi. Pertanto, eventuali modifiche alla procedura sono da apportare una volta sola.

7.2 Moduli principali

Il modulo che contiene la prima istruzione eseguibile di un programma è detto "modulo principale". A differenza di un modulo, una procedura non può contenere la prima istruzione da eseguire, perché l'esecuzione non può mai iniziare da una procedura.

Tutto ciò che è compreso in un modulo, tranne le procedure **SUB** e **FUNCTION**, è detto "codice a livello di modulo". In QuickBASIC, per codice a livello di modulo si intende ogni istruzione alla quale si può accedere senza passare ad una finestra di modifica delle procedure. Nell'illustrazione 7.1 si vede la relazione tra questi elementi.

Illustrazione 7.1 Modulo principale contenente codice a livello di modulo e procedure

7.3 Moduli contenenti solo procedure

Un modulo può essere costituito solo da procedure **SUB** e **FUNCTION** senza necessariamente contenere del codice a livello di modulo. Questo tipo di modulo è tra i più importanti.

I moduli contenenti solo procedure sono spesso usati per "separare" le procedure dal corpo del programma, il che facilita, ad esempio, la spartizione di un progetto di programma tra diversi programmatori. Inoltre, le procedure create per uso generale in vari programmi (quali quelle che valutano matrici, inviano dati binari ad una porta **COM**, modificano stringhe, o gestiscono errori), possono essere memorizzate nei moduli e quindi utilizzate in nuovi programmi semplicemente caricando in QuickBASIC il modulo appropriato.

Nota Se una procedura in un modulo contenente solo procedure richiede una routine di gestione degli errori o degli eventi, oppure un'istruzione **COMMON SHARED**, queste vanno inserite a livello di modulo.

7.4 Creazione di un modulo di sole procedure

E' facile creare un modulo contenente solo procedure, sia inserendo nuove procedure in un file separato, sia spostandone da un file ad un altro.

Per creare un nuovo modulo:

1. Richiamare QuickBASIC senza aprire o caricare alcun file.
2. Scrivere tutte le procedure **SUB** e **FUNCTION** che si desiderano, ma non immettere alcun codice a livello di modulo. (Fanno eccezione le routine di gestione degli errori o degli eventi e le dichiarazioni BASIC necessarie.)
3. Utilizzando il comando **Salva con nome**, assegnare un nome al modulo e memorizzarlo.

Per spostare le procedure da un modulo ad un altro:

1. Caricare i file contenenti le procedure che si desiderano spostare.
2. Se esiste già il file di destinazione, utilizzare il comando **Carica file** del menu **File** per caricarlo. Se non esiste, utilizzare il comando **Crea file** del menu **File** per crearne uno nuovo.
3. Selezionare il comando **SUBroutine** del menu **Visualizza** ed utilizzare l'opzione **Sposta** per trasferire le procedure dal file vecchio a quello nuovo. Questo passaggio si finalizza nel momento in cui si esce da QuickBASIC e si sceglie **Sì** nella finestra di dialogo che chiede se si vogliono memorizzare i file modificati; altrimenti le procedure ritornano nel punto in cui si trovavano all'inizio dell'esecuzione.

7.5 Caricamento di moduli

Utilizzando il comando **Carica file** del menu **File**, si possono caricare quanti moduli si desiderano (l'unica limitazione è la quantità di memoria disponibile). Tutte le procedure dei moduli caricati possono essere chiamate da qualunque altra procedura o dal codice a livello di modulo. Un modulo può tranquillamente contenere una procedura che non viene mai chiamata.

Alcuni o tutti i moduli caricati possono contenere del codice a livello di modulo. In QuickBASIC di solito l'esecuzione comincia nel codice a livello di modulo del primo modulo caricato. E' possibile, tuttavia, cominciare l'esecuzione in un modulo diverso, utilizzando il comando **Imposta il modulo principale** del menu **Esegui**. L'arresto dell'esecuzione, invece, avviene generalmente alla fine del modulo principale, e QuickBASIC non continua di proposito l'esecuzione nel codice a livello di modulo di altri moduli.

La possibilità di scegliere quale codice a livello di modulo verrà eseguito risulta particolarmente utile quando si confrontano due versioni dello stesso programma. Ad esempio, si possono provare diverse interfacce utente, inserendo ciascuna di esse in un modulo diverso. E' anche possibile collocare del codice di prova in un modulo contenente solo procedure e utilizzare, quindi, il comando **Imposta il modulo principale** per alternare tra il programma e i test.

Non è necessario tenersi al corrente dei moduli utilizzati dal programma. Ogni volta che si utilizza il comando **Salva tutto**, QuickBASIC crea, o aggiorna, un file .MAK che fornisce un elenco di tutti i moduli attualmente in memoria. La volta successiva in cui il modulo principale viene caricato con il comando **Apri programma**, QuickBASIC consulta questo file .MAK e carica automaticamente i moduli in esso elencati.

7.6 Utilizzo dell'istruzione DECLARE con moduli multipli

L'istruzione **DECLARE** ha diverse importanti funzioni. Utilizzando l'istruzione **DECLARE** si potrà:

- Specificare la sequenza e i tipi di dati dei parametri di una procedura.
- Attivare la verifica del tipo, che confermerà, ogni volta che si chiamerà una procedura, la corrispondenza tra argomenti e parametri per quanto riguarda il numero e il tipo di dati.
- Identificare il nome di una procedura **FUNCTION** come nome di una procedura e non di una variabile.
- Cosa importantissima, consentire al modulo principale di chiamare le procedure che si trovano in altri moduli o nelle librerie Quick.

QuickBASIC inserisce automaticamente nei moduli le istruzioni **DECLARE** richieste. La sezione 2.5.4, "Verifica degli argomenti con l'istruzione DECLARE", illustra le caratteristiche e i limiti di questo sistema.

Nonostante questo inserimento automatico, volendo, si può creare un file da includere a parte contenente tutte le istruzioni **DECLARE** che un programma richiede. Se poi si aggiungono o eliminano procedure o se ne modificano gli elenchi degli argomenti, aggiornare manualmente questo file.

Qualora i programmi vengano scritti con un editor di testo (piuttosto che nell'ambiente di programmazione QuickBASIC) e compilati con BC, le istruzioni **DECLARE** devono essere inserite manualmente.

7.7 Accesso alle variabili da due o più moduli

L'attributo **SHARED** permette l'accesso alle variabili sia a livello di modulo che all'interno delle procedure di quel modulo. Se una procedura viene spostata ad un altro modulo, però, le variabili non sono più condivise.

Le variabili si possono passare a ciascuna procedura attraverso il proprio elenco di argomenti. Ciò può risultare scomodo, tuttavia, se si deve passare un elevato numero di variabili.

Una soluzione consiste nell'utilizzare le istruzioni **COMMON**, che rendono possibile l'accesso di due o più moduli allo stesso insieme di variabili. La sezione 2.6, "Variabili condivise con SHARED", spiega le modalità dell'operazione.

Come alternativa, si può utilizzare un'istruzione **TYPE...END TYPE** per unire in una sola struttura di record tutte le variabili da passare. Negli elenchi degli argomenti e dei parametri, quindi, comparirà un nome di variabile soltanto, indipendentemente dal numero delle variabili passate.

Se si sta semplicemente scomponendo un programma e le sue procedure in moduli separati, risultano efficaci ambedue le alternative. Se invece si sta aggiungendo una procedura ad un modulo (per poi utilizzarla in altri programmi), si deve evitare l'utilizzo di un'istruzione **COMMON**. Mentre infatti i moduli dovrebbero semplificare l'inclusione di procedure già esistenti in nuovi programmi, le istruzioni **COMMON** complicano il processo. Se una procedura ha bisogno di un gran numero di variabili, forse non è da collocare in un modulo a parte.

7.8 Utilizzo di moduli durante lo sviluppo del programma

Quando si inizia un progetto di programmazione, è utile verificare se all'interno di moduli già esistenti vi siano procedure che possano essere riutilizzate per il nuovo software. Se qualcuna di queste procedure non si trova in un modulo a parte, sarà opportuno collocarvela.

Man mano che il programma prende forma, le procedure appena scritte entrano automaticamente a far parte del modulo principale. Esse possono essere spostate in un modulo a parte e testate, oppure aggiunte ad uno dei moduli insieme ad altre procedure create per uso generale e utilizzate in altri programmi.

Il programma potrebbe richiedere procedure scritte in altri linguaggi. (Ad esempio, il MASM è ideale per un'interfaccia diretta con l'hardware, il FORTRAN dispone di quasi tutte le funzioni matematiche, il Pascal consente la creazione di sofisticate strutture di dati, ed il C fornisce sia codici strutturati che l'accesso diretto alla memoria.) Per utilizzare tali procedure nel programma, compilarle ed eseguire il link ad una libreria Quick. Si può inoltre scrivere un modulo a parte per provare le procedure della libreria Quick nello stesso modo in cui si provano le altre.

7.9 Compilazione e collegamento di moduli

Il risultato di un progetto di programmazione è di solito un file .EXE autonomo. Questo può essere creato in QuickBASIC caricandone tutti i moduli e selezionando quindi il comando **Crea un file EXE** del menu **Esegui**.

Si possono inoltre compilare i moduli con il compilatore BC dalla riga di comando, e poi utilizzare LINK per unire i codici oggetto. I file di codice oggetto scritti in altri linguaggi possono essere sottoposti al linker contemporaneamente.

Nota Utilizzando il comando **Crea un file EXE**, tutto il codice a livello di modulo e ogni procedura attualmente in memoria vengono inclusi nel file .EXE, che il programma utilizzi o meno tale codice. Per rendere il programma il più compatto possibile, bisogna chiudere tutti i codici a livello di modulo e procedure superflue prima della compilazione. La stessa regola vale quando si utilizza il comando BC per compilare dalla riga di comando; inoltre, è opportuno eliminare dai file tutto il codice non utilizzato.

7.10 Librerie Quick

Sebbene le librerie Quick non siano moduli, è importante tenere presente la loro relazione con i moduli.

Una libreria Quick contiene solo procedure. Esse possono essere scritte non solo in QuickBASIC, ma anche in altri linguaggi Microsoft (C, Pascal, FORTRAN e MASM).

Una libreria Quick contiene solo codice compilato (i moduli contengono il codice sorgente di QuickBASIC). Essa è creata mediante il linkaggio (collegamento) di codici oggetto compilati (file .OBJ). Il codice di una libreria Quick può essere prodotto da qualsiasi combinazione di linguaggi Microsoft. L'appendice H, "Creazione ed utilizzo delle librerie Quick", spiega come creare le libreria Quick dal codice oggetto e come aggiungere del nuovo codice oggetto a librerie Quick già esistenti.

Le librerie Quick hanno diversi impieghi:

- Forniscono un'interfaccia tra QuickBASIC e altri linguaggi.
- Permettono ai progettisti di nascondere il software di proprietà esclusiva. Aggiornamenti e programmi di utilità possono essere distribuiti come librerie Quick senza rivelarne il codice di proprietà esclusiva.
- Vengono caricate più velocemente e sono di solito più piccole dei moduli. E' possibile velocizzare il processo alquanto lento di caricamento di un programma a più moduli, convertendo i moduli secondari in una libreria Quick.

Notare, tuttavia, che i moduli facilitano lo sviluppo delle procedure, perché subito pronti all'esecuzione dopo ogni modifica; non c'è bisogno di ricreare una libreria Quick. Per inserire le procedure QuickBASIC in una libreria, è buona norma attendere che le procedure siano complete ed interamente messe a punto.

Quando si crea una libreria Quick, tutto il codice a livello di modulo nel file da cui è stata creata viene automaticamente incluso. Tuttavia, dato che ad altri moduli questo codice non è accessibile, risulta solo uno spreco di spazio. Prima di convertire un modulo in libreria Quick, assicurarsi che siano state rimosse tutte le istruzioni a livello di modulo (eccetto qualsiasi gestore di errori o di eventi e le dichiarazioni utilizzate dalle procedure).

Nota Le librerie Quick non sono incluse nei file .MAK e devono essere caricate con l'opzione /L quando si avvia QuickBASIC. Una libreria Quick è un file con estensione .QLB creato contemporaneamente ad un altro file con estensione .LIB. Quest'ultimo contiene lo stesso codice della libreria Quick ma in una forma che ne consente l'aggancio al resto del programma per creare un'applicazione autonoma.

Se si utilizzano le librerie Quick per distribuire un codice di proprietà esclusiva (ad esempio, delle procedure per la manipolazione di dati), assicurarsi di includere i file .LIB, in modo che gli acquirenti possano creare applicazioni autonome che si servono di queste procedure. Altrimenti, si dovranno limitare ad eseguire le applicazioni dall'interno dell'ambiente QuickBASIC.

7.10.1 Creazione di librerie Quick

Utilizzando il comando **Crea libreria** del menu **Esegui** si crea una libreria Quick di procedure QuickBASIC contenente tutte le procedure residenti al momento, chiamate o meno dal programma, e tutto il codice a livello di modulo. Perché la libreria Quick sia compatta, è necessario rimuovere tutte le procedure inutilizzate e tutto il codice a livello di modulo superfluo prima di crearla.

Il numero delle librerie Quick è illimitato ed esse possono contenere qualunque combinazione di procedure. Tuttavia, in QuickBASIC si può caricare solo una libreria Quick alla volta. (Generalmente, si creano librerie Quick per applicazioni specifiche, contenenti soltanto le procedure necessarie ad un programma particolare.) Caricando molti moduli ed utilizzando il comando **Crea libreria**, si possono ottenere librerie Quick estese.

Una libreria Quick si può anche creare compilando uno o più moduli con il comando BC e sottoponendo al linker i file di codice oggetto. Allo stesso modo si ottengono librerie Quick di procedure scritte in altri linguaggi. Durante il linkaggio non è necessario limitarsi ad un solo linguaggio, poiché all'interno di una libreria Quick è possibile unire file di codice oggetto scritti in qualunque linguaggio Microsoft. L'appendice H, "Creazione ed utilizzo delle librerie Quick", spiega come convertire i file di codice oggetto (.OBJ) in librerie Quick.

7.11 Consigli per una buona programmazione a moduli

I moduli sono da utilizzare ai fini di una programmazione e di un'organizzazione del lavoro migliori. Questi suggerimenti possono essere d'aiuto.

- Riflettere ed organizzarsi prima di iniziare.

  Prima di iniziare un progetto è bene fare un elenco di tutte le operazioni che si vogliono far eseguire da procedure. Quindi, controllare se all'interno del proprio parco procedure ne esiste già qualcuna utilizzabile direttamente o con piccole modifiche. Evitare di ripetere inutilmente lavoro già fatto.

- Scrivere procedure di uso e applicazione generale.

  Le procedure è bene siano utilizzabili in diversi programmi, ma non per questo inutilmente complesse. Una buona procedura, quanto più è semplice tanto più è efficace.

  Risulta talvolta utile modificare una procedura già esistente per utilizzarla in un nuovo programma. Anche se ciò richiedesse di modificare programmi già scritti, ne varrà la pena se la nuova procedura avrà poi più ampia applicazione.

continua

- Quando si creano propri moduli di procedure, porre procedure logicamente distinte in moduli diversi.

  E' opportuno per esempio porre le procedure per la manipolazione delle stringhe in un modulo, quelle per la gestione delle matrici in un altro, e quelle per la comunicazione dei dati in un terzo. Questa sistemazione evita confusione e facilita il futuro reperimento delle procedure.

Parte seconda:
Gli elementi essenziali del BASIC

Parte seconda

Istruzioni IF...THEN...ELSE

Sintassi 1 (monoriga)

Sintassi 2 (a blocchi)

Gli elementi essenziali del BASIC

La seconda parte fornisce una guida di riferimento rapido a tutte le istruzioni e funzioni utilizzate in questa versione del BASIC.

Il capitolo 8 – presentato in ordine alfabetico in base alla parola chiave – fa un breve riassunto dell'azione di ciascuna istruzione o funzione e presenta, con alcune righe di sintassi, la forma corretta dell'istruzione. Ciò sarà particolarmente utile come riferimento durante la programmazione.

Il capitolo 9 è costituito da tabelle di riferimento, che raggruppano le istruzioni e le funzioni comunemente usate in base all'argomento di programmazione. Queste tabelle rispettano la stessa struttura dei primi sei capitoli del manuale. Esse possono servire a scoprire metodi alternativi per raggiungere particolari scopi di programmazione.

Capitoli

8 Panoramica delle istruzioni e funzioni

Questo capitolo offre un riassunto delle istruzioni e funzioni del QuickBASIC, facendo seguire ad ogni parola chiave una descrizione della funzione e una riga di sintassi, che mostra esattamente ciò che bisogna digitare per utilizzare l'istruzione.

Nell'introduzione di questo manuale sono descritte le convenzioni tipografiche utilizzate nelle righe di sintassi. Generalmente, le voci in grassetto sono da digitare esattamente come scritte nel manuale; i parametri formali indicanti informazioni da fornire sono in corsivo. Le parentesi quadre indicano voci facoltative.

Il Consulente QB (una guida in linea al linguaggio QuickBASIC) fornisce per ciascuna istruzione o funzione:

- Un riassunto dell'azione e della sintassi dell'istruzione (lo SchermoQuick)
- Informazioni complete sull'utilizzo dell'istruzione, inclusa la spiegazione dei parametri formali
- Uno o più esempi di programma che illustrano l'utilizzo dell'istruzione

Per accedere a queste informazioni, posizionare il cursore su qualunque parola chiave del BASIC che appare nella finestra di visualizzazione di QuickBASIC, e premere F1.

Le istruzioni e le funzioni di questo capitolo sono raggruppate per argomento nel capitolo 9, "Tabelle di riferimento rapido". Consultare questo capitolo per individuare le istruzioni da utilizzare per un particolare processo di programmazione.

Funzione ABS

Restituisce il valore assoluto di un'espressione numerica.

Sintassi **ABS** (*espressione-numerica*)

Funzione ASC

Restituisce il valore numerico corrispondente al codice ASCII del primo carattere di un'espressione a stringa.

Sintassi **ASC** (*espressionestringa*)

Funzione ATN

Restituisce l'arcotangente di un'espressione numerica (l'angolo la cui tangente è uguale all'espressione numerica).

Sintassi **ATN** (*espressione-numerica*)

Istruzione BEEP

Fa emettere un suono all'altoparlante.

Sintassi **BEEP**

Istruzione BLOAD

Carica un file di immagine della memoria, creato con **BSAVE**, da un file o da una periferica di input.

Sintassi **BLOAD** *specfile* [, *offset*]

Istruzione BSAVE

Trasferisce il contenuto di un'area di memoria ad un file o una periferica di output.

Sintassi **BSAVE** *specfile*, *offset*, *lunghezza*

Istruzione CALL (procedure BASIC)

Trasferisce il controllo ad una SUB del BASIC.

Sintassi 1 CALL *nome* [(*elencoargomenti*)]

Sintassi 2 *nome* [*elencoargomenti*]

Istruzione CALL, CALLS (procedure non BASIC)

Trasferisce il controllo ad una procedura scritta in un altro linguaggio.

Sintassi 1 CALL *nome* [(*elencoargomenti-call*)]

Sintassi 2 *nome* [*elencoargomenti-call*]

Sintassi 3 CALLS *nome* [(*elencoargomenti-calls*)]

Istruzione CALL ABSOLUTE

Trasferisce il controllo ad una procedura in linguaggio macchina.

Sintassi CALL ABSOLUTE ([*elencoargomenti*,] *variabileintera*)

Istruzioni CALL INT86OLD

Consentono al programma di eseguire chiamate di sistema DOS.

Sintassi CALL INT86OLD (*int_no*, *in_matrice* ( ), *out_matrice* ( ))
CALL INT86XOLD (*int_no*, *in_matrice* ( ), *out_matrice* ( ))

Istruzione CALL INTERRUPT

Consente ai programmi BASIC di eseguire chiamate di sistema DOS.

Sintassi CALL INTERRUPT (*numinterrupt*, *regprima*, *regdopo*)
CALL INTERRUPTX (*numinterrupt*, *regprima*, *regdopo*)

Funzione CDBL

Converte un'espressione numerica in un numero in doppia precisione.

Sintassi CDBL (*espressione-numerica*)

Istruzione CHAIN

Trasferisce il controllo dal programma corrente ad un altro.

Sintassi **CHAIN** *specfile*

Istruzione CHDIR

Cambia la directory corrente predefinita dell'unità disco specificata.

Sintassi **CHDIR** *specpercorso*

Funzione CHR$

Restituisce una stringa di un solo carattere il cui codice ASCII corrisponde all'argomento.

Sintassi **CHR$** (*codice*)

Funzione CINT

Converte un'espressione numerica in un intero arrotondando la parte decimale dell'espressione.

Sintassi **CINT** (*espressione-numerica*)

Istruzione CIRCLE

Disegna un'ellisse o un cerchio di centro e raggio specificati.

Sintassi **CIRCLE [STEP]** (*x*, *y*), *raggio* [, [*colore*] [, [*inizio*] [, [*fine*] [, *dimensione*]]]]

Istruzione CLEAR

Reinizializza tutte le variabili del programma, chiude i file, e imposta la dimensione dello stack.

Sintassi **CLEAR [, ,** *stack*]

Funzione CLNG

Converte un'espressione numerica in un intero lungo (a 4 byte) arrotondando la parte decimale dell'espressione.

Sintassi **CLNG** (*espressione-numerica*)

Istruzione CLOSE

Termina l'I/O su file o su periferica.

Sintassi **CLOSE** [[#]*numerofile* [, [#]*numerofile*]...]

Istruzione CLS

Cancella lo schermo.

Sintassi **CLS** [{**0** | **1** | **2**}]

Istruzione COLOR

Seleziona i colori dello schermo.

Sintassi	**COLOR** [*primopiano*] [, [*sfondo*] [*limite*]]	Modalità schermo 0
	COLOR [*sfondo*] [, *palette*]	Modalità schermo 1
	COLOR [*primopiano*] [, *sfondo*]	Modalità schermo 7–10
	COLOR [*primopiano*]	Modalità schermo 12–13

Istruzioni COM

COM ON attiva la gestione degli eventi di comunicazione sulla porta specificata, **COM OFF** la disattiva, e **COM STOP** la sospende.

Sintassi **COM** (*n*) **ON**
COM (*n*) **OFF**
COM (*n*) **STOP**

Funzione COMMAND$

Restituisce la riga di comando utilizzata per richiamare il programma.

Sintassi **COMMAND$**

Istruzione COMMON

Definisce le variabili globali da condividere tra moduli o per la concatenazione ad un altro programma.

Sintassi **COMMON [SHARED]** [/*nomeblocco*/] *elencovariabili*

Istruzione CONST

Dichiara le costanti simboliche da utilizzare al posto di valori numerici o di stringa.

Sintassi **CONST** *nomecostante* = *espressione* [, *nomecostante* = *espressione*]...

Funzione COS

Restituisce il coseno di un angolo espresso in radianti.

Sintassi **COS** (*espressione-numerica*)

Funzione CSNG

Converte un'espressione numerica in un valore in precisione semplice.

Sintassi **CSNG** (*espressione-numerica*)

Funzione CSRLIN

Restituisce la posizione di riga attuale del cursore.

Sintassi **CSRLIN**

Funzioni CVI, CVS, CVL, CVD

Convertono in numeri stringhe contenenti valori numerici.

Sintassi **CVI** (*stringa-a-2-byte*)
CVS (*stringa-a-4-byte*)
CVL (*stringa-a-4-byte*)
CVD (*stringa-a-8-byte*)

Funzioni CVSMBF, CVDMBF

Convertono stringhe contenenti numeri in formato Binario Microsoft in numeri in formato IEEE.

Sintassi **CVSMBF** (*stringa-a-4-byte*)
CVDMBF (*stringa-a-8-byte*)

Istruzione DATA

Contiene le costanti numeriche e di stringa utilizzate dalle istruzioni **READ** di un programma.

Sintassi **DATA** *costante1* [, *costante2*]

Funzione DATE$

Restituisce una stringa contenente la data corrente.

Sintassi **DATE$**

Istruzione DATE$

Imposta la data corrente.

Sintassi **DATE$** = *espressionestringa*

Istruzione DECLARE (procedure BASIC)

Dichiara i riferimenti alle procedure BASIC e attiva la verifica del tipo degli argomenti.

Sintassi **DECLARE {FUNCTION | SUB}** *nome* [([*elencoparametri*])]

Istruzione DECLARE (procedure non BASIC)

Dichiara sequenze di chiamata per procedure esterne scritte in altri linguaggi.

Sintassi 1 **DECLARE FUNCTION** *nome* [CDECL] [ALIAS "*sinonimo*"] [([*elencoparametri*])]

Sintassi 2 **DECLARE SUB** *nome* [CDECL] [ALIAS "*sinonimo*"] [([*elencoparametri*])]

Istruzione DEF FN

Definisce ed assegna un nome ad una funzione.

Sintassi 1 **DEF FN***nome* [(*elencoparametri*)] = *espressione*

Sintassi 2 **DEF FN***nome* [(*elencoparametri*)]
.
.
.
FN*nome* = *espressione*
.
.
.
END DEF

Istruzione DEF SEG

Imposta l'indirizzo corrente del segmento per una successiva funzione **PEEK** o istruzione **BLOAD**, **BSAVE**, **CALL**, **ABSOLUTE** o **POKE**.

Sintassi **DEF SEG** [= *indirizzo*]

Istruzioni DEF*tipo*

Impostano il tipo di dati predefinito delle variabili, delle funzioni **DEF FN**, e delle procedure **FUNCTION**.

Sintassi **DEFINT** *intervallolettere* [, *intervallolettere*]...
DEFSNG *intervallolettere* [, *intervallolettere*]...
DEFDBL *intervallolettere* [, *intervallolettere*]...
DEFLNG *intervallolettere* [, *intervallolettere*]...
DEFSTR *intervallolettere* [, *intervallolettere*]...

Istruzione DIM

Dichiara una variabile e le assegna spazio in memoria.

Sintassi **DIM** [**SHARED**] *variabile* [(*indici*)] [**AS** *tipo*] [, *variabile* [(*indici*)] [**AS** *tipo*]...

Istruzioni DO...LOOP

Ripetono un blocco di istruzioni fintantoché una condizione è vera oppure falsa.

Sintassi 1 **DO**
[*bloccoistruzioni*]
LOOP [{WHILE | UNTIL} *espressionebooleana*]

Sintassi 2 **DO [{WHILE | UNTIL}** *espressionebooleana*]
[*bloccoistruzioni*]
LOOP

Istruzione DRAW

Disegna un oggetto definito da *espressionestringa*.

Sintassi **DRAW** *espressionestringa*

Istruzione END

Termina un programma BASIC, una procedura, o un'istruzione a blocchi.

Sintassi **END [{DEF | FUNCTION | IF | SELECT | SUB | TYPE}]**

Funzione ENVIRON$

Estrae una stringa di ambiente dalla tabella delle stringhe di ambiente DOS.

Sintassi **ENVIRON$** (*stringambiente*)
ENVIRON$ (*n*)

Istruzione ENVIRON

Modifica un parametro nella tabella delle stringhe di ambiente DOS.

Sintassi **ENVIRON** *espressionestringa*

Funzione EOF

Prova la condizione di fine file.

Sintassi **EOF** (*numerofile*)

Istruzione ERASE

Reinizializza gli elementi delle matrici statiche e disalloca le matrici dinamiche.

Sintassi **ERASE** *nomematrice* **[**, *nomematrice*...**]**

Funzioni ERDEV, ERDEV$

Forniscono informazioni sulla condizione di una determinata periferica dopo un errore.

Sintassi **ERDEV**
ERDEV$

Funzioni ERR, ERL

Restituiscono informazioni sugli errori.

Sintassi **ERR**
ERL

Istruzione ERROR

Simula la presenza di un errore BASIC e consente all'utente di definire dei codici di errore.

Sintassi **ERROR** *espressioneintera*

Istruzione EXIT

Fa uscire da una funzione **DEF FN**, da un ciclo **DO...LOOP** o **FOR...NEXT**, da una **FUNCTION** o da una procedura **SUB**.

Sintassi **EXIT {DEF | DO | FOR | FUNCTION | SUB}**

Funzione EXP

Calcola la funzione esponenziale.

Sintassi **EXP** (*x*)

Istruzione FIELD

Assegna spazio alle variabili nel buffer di un file ad accesso casuale.

Sintassi **FIELD [#]***numerofile*, *ampiezzacampo* **AS** *variabilestringa*...

Funzione FILEATTR

Restituisce informazioni relative ad un file aperto.

Sintassi **FILEATTR** (*numerofile*, *attributo*)

Istruzione FILES

Visualizza i nomi dei file residenti sul disco specificato.

Sintassi **FILES** [*specfile*]

Funzione FIX

Restituisce la parte intera troncata di *x*.

Sintassi **FIX** (*x*)

Istruzione FOR...NEXT

Ripete un gruppo di istruzioni un determinato numero di volte.

Sintassi **FOR** *contatore* = *inizio* **TO** *fine* [**STEP** *incremento*]
.
.
.
NEXT [*contatore* [, *contatore*...]]

Funzione FRE

Restituisce la quantità di spazio memoria disponibile.

Sintassi 1 **FRE** (*espressionenumerica*)

Sintassi 2 **FRE** (*espressionestringa*)

Funzione FREEFILE

Restituisce il successivo numero di file BASIC libero.

Sintassi **FREEFILE**

Istruzione FUNCTION

Dichiara il nome, i parametri, e il codice che costituiscono la parte principale di una procedura **FUNCTION**.

Sintassi **FUNCTION** *nome* [(*listaparametri*)] [**STATIC**]

.
.
.
nome = *espressione*
.
.
.
END FUNCTION

Istruzione GET – I/O su file

Assegna a un buffer ad accesso casuale o a una variabile i dati letti da un file.

Sintassi **GET** [**#**]*numerofile* [, [*numerorecord*] [, *variabile*]

Istruzione GET – grafica

Memorizza immagini grafiche sullo schermo.

Sintassi **GET** [**STEP**](*x1*, *y1*)–[**STEP**](*x2*, *y2*), *nomematrice* [(*indici*)]

Istruzione GOSUB...RETURN

Salta a, e ritorna da, una subroutine.

Sintassi **GOSUB** {*etichettariga1* | *numeroriga1*}

.
.
.
RETURN [*etichettariga2* | *numeroriga2*]

Istruzione GOTO

Salta incondizionatamente alla riga specificata.

Sintassi **GOTO** {*etichettariga* | *numeroriga*}

Funzione HEX$

Restituisce come stringa il formato esadecimale dell'argomento decimale *espressione*.

Sintassi **HEX$** (*espressione*)

Istruzioni IF...THEN...ELSE

Permettono un'esecuzione condizionata al valore di un'espressione booleana.

Sintassi 1 (monoriga) **IF** *espressionebooleana* **THEN** *partedithen*
[**ELSE** *partedielse*]

Sintassi 2 (a blocchi) **IF** *espressionebooleana1* **THEN**
[*bloccoistruzioni-1*]
[**ELSEIF** *espressionebooleana2* **THEN**
[*bloccoistruzioni-2*]]
.
.
.
[**ELSE**
[*bloccoistruzioni-n*]]
END IF

Funzione INKEY$

Legge un carattere dalla tastiera.

Sintassi **INKEY$**

Funzione INP

Restituisce il byte letto da una porta di I/O.

Sintassi **INP** (*porta*)

Funzione INPUT$

Restituisce una stringa di caratteri letta dal file specificato.

Sintassi **INPUT$** (*n* [, [#]*numerofile*])

Istruzione INPUT

Permette l'input dalla tastiera durante l'esecuzione del programma.

Sintassi **INPUT** [;] ["*stringaprompt*" { ; | , }] *elencovariabili*

Istruzione INPUT

Legge i dati da una periferica o file sequenziale e li assegna alle variabili.

Sintassi **INPUT #***numerofile*, *elencovariabili*

Funzione INSTR

Restituisce la posizione del carattere della prima occorrenza di una stringa in un'altra.

Sintassi **INSTR** ([*inizio*,] *espressionestringa1*, *espressionestringa2*)

Funzione INT

Restituisce l'intero più grande minore o uguale a *espressione-numerica*.

Sintassi **INT** (*espressione-numerica*)

Funzione IOCTL$

Riceve una stringa di controllo dei dati da un driver di periferica.

Sintassi **IOCTL$** ([#] *numerofile*)

Istruzione IOCTL

Trasmette una stringa di controllo dei dati ad un driver di periferica.

Sintassi **IOCTL** [#] *numerofile*, *stringa*

Istruzioni KEY

Assegnano valori programmati ai tasti funzione, poi ne visualizzano i valori e attivano o disattivano la riga di visualizzazione dei tasti FUNZIONE.

Sintassi **KEY** *n*, *espressionestringa*
KEY LIST
KEY ON
KEY OFF

Istruzioni KEY (*n*)

KEY ON attiva la gestione dei tasti specificati, **KEY OFF** la disattiva, e **KEY STOP** la sospende.

Sintassi **KEY** (*n*) **ON**
KEY (*n*) **OFF**
KEY (*n*) **STOP**

Istruzione KILL

Elimina un file dal disco.

Sintassi **KILL** *specfile*

Funzione LBOUND

Restituisce il limite inferiore (il più piccolo indice disponibile) della dimensione indicata di una matrice.

Sintassi **LBOUND** (*matrice*) [, *dimensione*]

Funzione LCASE$

Restituisce un'espressione a stringa in caratteri minuscoli.

Sintassi **LCASE$** (*espressionestringa*)

Funzione LEFT$

Restituisce una stringa consistente degli *n* caratteri all'estrema sinistra di una stringa.

Sintassi **LEFT$** (*espressionestringa*, *n*)

Funzione LEN

Restituisce il numero di caratteri di una stringa o il numero di byte richiesti da una variabile.

Sintassi **LEN** (*espressionestringa*)
LEN (*variabile*)

Istruzione LET

Assegna il valore di un'espressione ad una variabile.

Sintassi [**LET**] *variabile* = *espressione*

Istruzione LINE

Disegna una linea o un riquadro sullo schermo.

Sintassi **LINE** [[**STEP**] (*x1*, *y1*)]-[**STEP**] (*x2*, *y2*) [, [*colore*] [, [**B** [**F**]] [, *stile*]]]

Istruzione LINE INPUT

Immette una riga intera (fino a 255 caratteri) in una variabile a stringa, senza utilizzare delimitatori.

Sintassi **LINE INPUT** [;] ["*stringaprompt*";] *variabilestringa*

Istruzione LINE INPUT

Immette in una variabile a stringa una riga intera senza delimitatori letta da un file ad accesso sequenziale.

Sintassi **LINE INPUT #***numerofile*, *variabilestringa*

Funzione LOC

Restituisce la posizione corrente all'interno del file.

Sintassi **LOC** (*numerofile*)

Istruzione LOCATE

Sposta il cursore al punto specificato.

Sintassi **LOCATE** [*riga*] [, [*colonna*] [, [*cursore*] [, [*inizio*, *stop*]]]]

Istruzione LOCK...UNLOCK

Impedisce l'accesso di altri processi a parte o a tutto il file aperto.

Sintassi **LOCK** [#]*numerofile* [, {*record* | [*inizio*] **TO** *fine*}]

.

.

.

UNLOCK [#]*numerofile* [, {*record* | [*inizio*] **TO** *fine*}]

Funzione LOF

Restituisce la lunghezza in byte del file specificato.

Sintassi **LOF** (*numerofile*)

Funzione LOG

Restituisce il logaritmo naturale di una espressione numerica.

Sintassi **LOG** (*n*)

Funzione LPOS

Restituisce la posizione corrente della testina di scrittura della stampante all'interno del buffer della stampante.

Sintassi **LPOS** (*n*)

Istruzioni LPRINT, LPRINT USING

Stampano informazioni sulla stampante collegata alla porta LPT1.

Sintassi 1 **LPRINT** [*elencoespressioni*] [{ ; | , }]

Sintassi 2 **LPRINT USING** *stringaformato*; *elencoespressioni* [{ ; | , }]

Istruzione LSET

Sposta i dati dalla memoria al buffer di un file ad accesso casuale (in attesa di un'istruzione **PUT**), copia una variabile di record in un altra, o giustifica a sinistra il valore di una stringa in una variabile a stringa.

Sintassi **LSET** {*variabilestringa* = *espressionestringa* |
espressionestringa1 = *espressionestringa2*}

Funzione LTRIM$

Restituisce una copia priva di spazi iniziali della stringa argomento.

Sintassi **LTRIM$** (*espressionestringa*)

Funzione MID$

Restituisce una sottostringa di una stringa.

Sintassi **MID$** (*espressionestringa*, *inizio* [, *lunghezza*])

Istruzione MID$

Sostituisce una parte di una variabile a stringa con un'altra stringa.

Sintassi **MID$** (*variabilestringa*, *inizio* [, *lunghezza*]) = *espressionestringa*

Funzioni MKD$, MKI$, MKL$, MKS$

Convertono i valori numerici in valori a stringa.

Sintassi **MKI$** (*espressioneintera*)
MKS$ (*espressione-in-precisione-semplice*)
MKL$ (*espressione-ad-intero-lungo*)
MKD$ (*espressione-in-precisione-doppia*)

Istruzione MKDIR

Crea una nuova directory.

Sintassi **MKDIR** *nomepercorso*

Funzioni MKSMBF$, MKDMBF$

Convertono numeri in formato IEEE in stringhe contenenti numeri in formato Binario Microsoft.

Sintassi **MKSMBF$** (*espressione-in-precisione-semplice*)
MKSMBF$ (*espressione-in-precisione-doppia*)

Istruzione NAME

Modifica il nome di un file o di una directory.

Sintassi **NAME** *vecchionomefile* **AS** *nuovonomefile*

Funzione OCT$

Restituisce come stringa il valore ottale dell'argomento numerico.

Sintassi **OCT$** (*espressione-numerica*)

Istruzioni ON *evento*

Indicano la prima riga di una subroutine di gestione degli eventi.

Sintassi **ON** *evento* **GOSUB** {*numeroriga* | *etichettariga*}

Istruzione ON ERROR

Attiva la gestione di errori e specifica la prima riga della routine di gestione degli errori.

Sintassi **ON ERROR GOTO** *riga*

Istruzione ON UEVENT GOSUB

Definisce il gestore per un evento definito dall'utente.

Sintassi **ON UEVENT GOSUB** {*numeroriga* | *etichettariga*}

Istruzioni ON...GOSUB, ON...GOTO

Saltano ad una delle righe specificate, a seconda del valore di un'espressione.

Sintassi 1 **ON** *espressione* **GOSUB** {*elenco-numeri-righe* | *elenco-etichette-righe*}

Sintassi 2 **ON** *espressione* **GOTO** {*elenco-numeri-righe* | *elenco-etichette-righe*}

Istruzione OPEN

Attiva l'I/O su un file o su una periferica.

Sintassi 1 **OPEN** *file* [**FOR** *modalità1*] [**ACCESS** *accesso*] [*blocco*] **AS** [#]*numfile* [**LEN** = *lungrec*]

Sintassi 2 **OPEN** *modalità2*, [#]*numfile*, *file* [, *lungrec*]

Istruzione OPEN COM

Apre ed inizializza una porta di comunicazioni per operazioni di I/O.

Sintassi **OPEN "COM***n*: *elencoopz1 elencoopz2***"** [**FOR** *modalità*] **AS** [#]*numfile* [**LEN** = *lungrec*]

Istruzione OPTION BASE

Dichiara il limite inferiore predefinito per gli indici di una matrice.

Sintassi **OPTION BASE** *n*

Istruzione OUT

Invia un byte ad una porta di I/O del computer.

Sintassi **OUT** *porta*, *dato*

Istruzione PAINT

Riempie un'area grafica con il colore o il motivo specificato.

Sintassi **PAINT** [**STEP**] (*x*, *y*) [, [*paint*] [, [*colorebordo*] [, *sfondo*]]]

Istruzioni PALETTE, PALETTE USING

Modifica uno o più colori della palette.

Sintassi **PALETTE** [*attributo*, *colore*]
PALETTE USING *nomematrice* [(*indicematrice*)]

Istruzione PCOPY

Copia una pagina schermo in un'altra.

Sintassi **PCOPY** *paginaorigine*, *paginadestinazione*

Funzione PEEK

Restituisce il byte memorizzato in una determinata posizione di memoria.

Sintassi **PEEK** (*indirizzo*)

Funzione PEN

Legge le coordinate della penna ottica.

Sintassi **PEN** (*n*)

Istruzioni PEN ON, OFF e STOP

PEN ON attiva la gestione degli eventi relativi alla penna ottica, **PEN OFF** la disattiva, e **PEN STOP** la sospende.

Sintassi **PEN ON**
PEN OFF
PEN STOP

Funzione PLAY

Restituisce il numero di note che si trovano ancora nella coda della musica di sottofondo.

Sintassi **PLAY** (*n*)

Istruzione PLAY

Esegue un brano musicale come specificato da una stringa.

Sintassi **PLAY** *stringacomando*

Istruzioni PLAY ON, OFF, e STOP

PLAY ON attiva la gestione degli eventi di suono, **PLAY OFF** la disattiva, e **PLAY STOP** la sospende.

Sintassi **PLAY ON**
PLAY OFF
PLAY STOP

Funzione PMAP

Restituisce la coordinata fisica corrispondente alla coordinata logica *espressione*, oppure viceversa, a seconda dell'argomento *funzione*.

Sintassi **PMAP** (*espressione*, *funzione*)

Funzione POINT

Legge il numero del colore o restituisce le coordinate di un pixel sullo schermo.

Sintassi **POINT** (*x*, *y*)
POINT (*numero*)

Istruzione POKE

Scrive un byte nel punto specificato dello spazio memoria.

Sintassi **POKE** *indirizzo*, *byte*

Funzione POS

Restituisce la posizione orizzontale corrente del cursore.

Sintassi **POS (0)**

Istruzione PRESET

Traccia sullo schermo il punto specificato.

Sintassi **PRESET [STEP]** (*coordinata-x*, *coordinata-y*) [, *colore*]

Istruzione PRINT

Visualizza dati sullo schermo.

Sintassi **PRINT** [*elencoespressioni*] [{ , | ; }]

Istruzioni PRINT #, PRINT # USING

Scrivono dati in un file ad accesso sequenziale.

Sintassi **PRINT #***numerofile*, **[USING** *espressionestringa*;**]** *elencoespressioni* [{ , | ; }]

Istruzione PRINT USING

Visualizza stringhe o numeri nel formato specificato.

Sintassi **PRINT USING** *stringaformato*; *elencoespressioni* [{ , | ; }]

Istruzione PSET

Disegna un punto sullo schermo.

Sintassi **PSET [STEP]** (*coordinata-x*, *coordinata-y*) [, *colore*]

Istruzione PUT – I/O su file

Scrive su un file i dati di una variabile o di un buffer ad accesso casuale.

Sintassi **PUT** [#]*numerofile* [, [*numerorecord*] [, *variabile*]]
PUT [#]*numerofile* [, {*numerorecord* | *numerorecord*, *variabile* | , *variabile*}]

Istruzione PUT – grafica

Visualizza sullo schermo un'immagine grafica ottenuta con un'istruzione **GET**.

Sintassi **PUT [STEP]** (*x*, *y*), *nomematrice* [(*indici*)] [, *verboazione*]

Istruzione RANDOMIZE

Inizializza (riseleziona) il generatore di numeri casuali.

Sintassi **RANDOMIZE** *elencovariabili*

Istruzione READ

Legge i valori da un'istruzione **DATA** e li assegna alle variabili.

Sintassi **READ** *elencovariabili*

Istruzione REDIM

Modifica lo spazio assegnato ad una matrice dichiarata **$DYNAMIC**.

Sintassi **REDIM [SHARED]** *variabile* (*indici*) **[AS** *tipo*] [, *variabile* (*indici*) **[AS** *tipo*]]...

Istruzione REM

Consente di inserire commenti esplicativi in un programma.

Sintassi 1 **REM** *commento*

Sintassi 2 ' *commento*

Istruzione RESET

Chiude tutti i file del disco.

Sintassi **RESET**

Istruzione RESTORE

Permette che le istruzioni **DATA** siano rilette dalla riga specificata.

Sintassi **RESTORE** [{*numeroriga* | *etichettariga*}]

Istruzione RESUME

Fa continuare l'esecuzione del programma dopo che è stata richiamata una routine di gestione degli errori.

Sintassi **RESUME [0]**
RESUME NEXT
RESUME {*numeroriga* | *etichettariga*}

Istruzione RETURN

Fa tornare il controllo da una subroutine.

Sintassi **RETURN** [{*numeroriga* | *etichettariga*}]

Funzione RIGHT$

Restituisce gli *n* caratteri all'estrema destra di una stringa.

Sintassi **RIGHT$** (*espressionestringa*, *n*)

Istruzione RMDIR

Elimina una directory esistente.

Sintassi **RMDIR** *nomepercorso*

Funzione RND

Restituisce un numero in precisione semplice a caso compreso tra 0 e 1.

Sintassi **RND** [(*n*)]

Istruzione RSET

Sposta dati dalla memoria al buffer di un file ad accesso casuale (in attesa di un'istruzione **PUT**) o giustifica a destra, in una variabile a stringa, il valore della stringa.

Sintassi **RSET** *variabilestringa* = *espressionestringa*

Funzione RTRIM$

Restituisce una stringa priva degli eventuali spazi vuoti finali.

Sintassi **RTRIM$** (*espressionestringa*)

Istruzione RUN

Riavvia il programma attualmente in memoria o esegue il programma specificato.

Sintassi **RUN** [{*numeroriga* | *rigacomando*}]

Funzione SADD

Restituisce l'indirizzo dell'espressione a stringa specificata.

Sintassi **SADD** (*variabilestringa*)

Funzione SCREEN

Legge il valore ASCII o il colore del carattere nel punto specificato dello schermo.

Sintassi **SCREEN** (*riga*, *colonna* [, *flagcolore*]

Istruzione SCREEN

Imposta le caratteristiche dello schermo.

Sintassi **SCREEN** [*modalità*] [, [*switchcolore*]] [, [*paginaa*]] [, [*paginav*]]

Funzione SEEK

Restituisce la posizione corrente nel file.

Sintassi **SEEK** (*numerofile*)

Istruzione SEEK

Imposta il punto per la successiva scrittura o lettura nel file.

Sintassi **SEEK** [#]*numerofile*, *posizione*

Istruzione SELECT CASE

Esegue uno dei blocchi di istruzioni elencati in base al valore di un'espressione.

Sintassi **SELECT CASE** *espressioneprova*
CASE *elencoespressioni1*
  [*bloccoistruzioni-1*]
[**CASE** *elencoespressioni2*]
  [*bloccoistruzioni-2*]
.
.
.
[**CASE ELSE**
  [*bloccoistruzioni-n*]]
END SELECT

Funzione SETMEM

Modifica la quantità di memoria utilizzata dallo spazio memoria di tipo FAR – l'area in cui vengono memorizzati gli oggetti FAR e le tabelle interne.

Sintassi **SETMEM** (*espressione-numerica*)

Funzione SGN

Indica il segno di un'espressione numerica.

Sintassi **SGN** (*espressione-numerica*)

Istruzione SHARED

Consente ad una procedura **SUB** o **FUNCTION** di accedere alle variabili dichiarate a livello di modulo, senza doverle passare come parametri.

Sintassi **SHARED** *variabile* [**AS** *tipo*] [, *variabile* [**AS** *tipo*]]...

Istruzione SHELL

Fa uscire dal programma BASIC, esegue un programma .COM, .EXE, .BAT o un comando DOS, e quindi ritorna nel programma alla riga successiva all'istruzione **SHELL**.

Sintassi **SHELL** [*stringacomando*]

Funzione SIN

Restituisce il seno dell'angolo *x*, dove *x* è dato in radianti.

Sintassi **SIN** (*x*)

Istruzione SLEEP

Sospende l'esecuzione del programma.

Sintassi **SLEEP** [*secondi*]

Istruzione SOUND

Genera un suono attraverso l'altoparlante.

Sintassi **SOUND** *frequenza*, *durata*

Funzione SPACE$

Restituisce una stringa costituita da *n* spazi.

Sintassi **SPACE$** (*n*)

Funzione SPC

Salta *n* spazi in un'istruzione **PRINT**.

Sintassi **SPC** (*n*)

Funzione SQR

Calcola la radice quadrata di *n*.

Sintassi **SQR** (*n*)

Istruzione STATIC

Rende locali rispetto ad una funzione **DEF FN**, ad una **FUNCTION**, o ad una **SUB** le variabili e le matrici semplici, e ne conserva i valori tra una chiamata e l'altra.

Sintassi **STATIC** *elencovariabili*

Funzione STICK

Restituisce le coordinate *x* e *y* dei due joystick.

Sintassi **STICK** (*n*)

Istruzione STOP

Termina l'esecuzione del programma.

Sintassi **STOP**

Funzione STR$

Restituisce una rappresentazione a stringa del valore di un'espressione numerica.

Sintassi **STR$** (*espressione-numerica*)

Funzione e istruzione STRIG

Restituiscono lo stato dello specificato pulsante d'azione del joystick.

Sintassi 1 (funzione) **STRIG** (*n*)

Sintassi 2 (istruzione) **STRIG {ON | OFF}**

Istruzioni STRIG ON, OFF, e STOP

STRIG ON attiva la gestione degli eventi del joystick, **STRIG OFF** la disattiva, e **STRIG STOP** la sospende.

Sintassi **STRIG** (*n*) **ON**
STRIG (*n*) **OFF**
STRIG (*n*) **STOP**

Funzione STRING$

Restituisce una stringa tutti i caratteri della quale hanno il codice ASCII specificato o corrispondono al primo carattere di un'espressione a stringa.

Sintassi **STRING$** (*m*, *n*)
STRING$ (*m*, *espressionestringa*)

Istruzioni SUB

Indicano l'inizio e la fine di un sottoprogramma.

Sintassi **SUB** *nomeglobale* [(*elencoparametri*)] [**STATIC**]
.
.
.
[**EXIT SUB**]
.
.
.
END SUB

Istruzione SWAP

Scambia i valori di due variabili.

Sintassi **SWAP** *variabile1*, *variabile2*

Istruzione SYSTEM

Chiude tutti i file aperti e restituisce il controllo al sistema operativo.

Sintassi **SYSTEM**

Funzione TAB

Sposta la posizione per la stampa.

Sintassi **TAB** (*colonna*)

Funzione TAN

Restituisce la tangente dell'angolo *x*, dove *x* è dato in radianti.

Sintassi **TAN** (*x*)

Funzione TIME$

Restituisce l'ora corrente, ottenuta dal sistema operativo.

Sintassi **TIME$**

Istruzione TIME$

Imposta l'ora.

Sintassi **TIME$** = *espressionestringa*

Funzione TIMER

Restituisce il numero dei secondi trascorsi dalla mezzanotte.

Sintassi **TIMER**

Istruzioni TIMER ON, OFF e STOP

TIMER ON attiva la gestione degli eventi del temporizzatore, **TIMER OFF** la disattiva, e **TIMER STOP** la sospende.

Sintassi **TIMER ON**
TIMER OFF
TIMER STOP

Istruzioni TRON, TROFF

TRON attiva, e **TROFF** disattiva, la traccia dell'esecuzione delle istruzioni del programma.

Sintassi **TRON**
TROFF

Istruzione TYPE

Definisce un tipo di dati contenente uno o più elementi.

Sintassi **TYPE** *tipoutente*
nomeelemento **AS** *nometipo*
nomeelemento **AS** *nometipo*
.
.
.
END TYPE

Funzione UBOUND

Restituisce il limite superiore (l'indice più grande disponibile) della dimensione di una matrice.

Sintassi **UBOUND** (*matrice*) [, *dimensione*])

Funzione UCASE$

Restituisce un'espressione a stringa con tutti i caratteri maiuscoli.

Sintassi **UCASE$** (*espressionestringa*)

Istruzione UEVENT

Attiva, disattiva o sospende la gestione di un evento definito dall'utente.

Sintassi **UEVENT ON**
UEVENT OFF
UEVENT STOP

Istruzione UNLOCK

Libera le parti di un file dai blocchi a cui erano state sottoposte.

Sintassi **UNLOCK** [#]*numerofile* [, {*record* | [*inizio*] **TO** *fine*}]

Funzione VAL

Restituisce il valore numerico di una stringa di cifre.

Sintassi **VAL** (*espressionestringa*)

Funzioni VARPTR, VARSEG

Restituiscono l'indirizzo di una variabile.

Sintassi **VARPTR** (*nomevariabile*)
VARSEG (*nomevariabile*)

Funzione VARPTR$

Restituisce una rappresentazione a stringa dell'indirizzo di una variabile, ad uso delle istruzioni **DRAW** e **PLAY**.

Sintassi **VARPTR$** (*nomevariabile*)

Istruzione VIEW

Definisce i limiti di schermo per l'output grafico.

Sintassi **VIEW** [[**SCREEN**] (*x1*, *y1*)–(*x2*, *y2*) [, [*colore*] [, *bordo*]]]

Istruzione VIEW PRINT

Imposta i limiti della finestra di testo dello schermo.

Sintassi **VIEW PRINT** [*rigasuperiore* **TO** *rigainferiore*]

Istruzione WAIT

Sospende l'esecuzione del programma durante il controllo dello stato di una porta di input del computer.

Sintassi **WAIT** *numeroporta*, *espressione-and* [, *espressione-xor*]

Istruzione WHILE...WEND

Esegue una serie di istruzioni in un ciclo mentre rimane vera una determinata condizione.

Sintassi **WHILE** *condizione*
.
.
.
[*istruzioni*]
.
.
.
WEND

Istruzione WIDTH

Imposta l'ampiezza della riga di output ad un file o ad una periferica, o modifica il numero di colonne e di righe visualizzate sullo schermo.

Sintassi **WIDTH** [*colonne*] [, *righe*]
WIDTH {#*numerofile* | *periferica*}, *ampiezza*
WIDTH LPRINT *ampiezza*

Istruzione WINDOW

Definisce le dimensioni logiche della finestra corrente.

Sintassi **WINDOW** [[**SCREEN**] (*x1*, *y1*)–(*x2*, *y2*)]

Istruzione WRITE

Invia dati allo schermo.

Sintassi **WRITE** [*elencoespressioni*]

Istruzione WRITE

Scrive dati in un file ad accesso sequenziale.

Sintassi **WRITE** #*numerofile*, *elencoespressioni*

9 Tabelle di riferimento rapido

Ogni sezione di questo capitolo espone brevemente un gruppo di istruzioni o funzioni generalmente utilizzate insieme. In ogni tabella viene elencato il tipo di processo eseguito, (per esempio un ciclo o una ricerca), il nome dell'istruzione o della funzione, e l'azione dell'istruzione.

Gli argomenti seguenti sono riassunti sotto forma di tabella:

- Istruzioni di controllo
- Istruzioni utilizzate in procedure BASIC
- Istruzioni di I/O standard
- Istruzioni di I/O su file
- Istruzioni e funzioni di manipolazione di stringhe
- Istruzioni e funzioni grafiche
- Istruzioni e funzioni di gestione

Queste tabelle possono essere utilizzate sia come guida di riferimento all'azione di ciascuna istruzione o funzione, sia per identificare istruzioni logicamente vicine.

9.1 Riassunto delle istruzioni di controllo

La tabella 9.1 elenca le istruzioni BASIC utilizzate per controllare l'ordine di esecuzione di un programma.

Tabella 9.1 Istruzioni utilizzate nei cicli e nelle decisioni

Processo	***Istruzione***	***Azione***
Esecuzione di un ciclo	**FOR...NEXT**	Ripete le istruzioni tra **FOR** e **NEXT** un determinato numero di volte.
	EXIT FOR	Fornisce un metodo alternativo per uscire da un ciclo **FOR...NEXT**.
	DO...LOOP	Ripete le istruzioni comprese tra **DO** e **LOOP**, finché non si avvera una data condizione (**DO...LOOP UNTIL** *condizione*), o purché rimanga vera una data condizione (**DO...LOOP WHILE** *condizione*).
	EXIT DO	Fornisce un metodo alternativo per uscire da un ciclo **DO...LOOP**.
	WHILE...WEND	Ripete le istruzioni comprese tra **WHILE** e **WEND** purché rimanga vera una data condizione (analogamente a **DO WHILE** *condizione***...LOOP**).
Processi di decisione	**IF...THEN...ELSE**	Esegue o salta ad istruzioni diverse in base alle condizioni.
	SELECT CASE	Esegue istruzioni diverse in base alle condizioni.

9.2 Riassunto delle istruzioni utilizzate nelle procedure BASIC

La tabella 9.2 elenca le istruzioni utilizzate in BASIC per definire, dichiarare, chiamare e passare argomenti alle procedure BASIC. Essa elenca, inoltre, le istruzioni che consentono di condividere le variabili tra procedure, moduli, e diversi programmi concatenati.

Tabella 9.2 Istruzioni utilizzate nelle procedure

Processo	*Istruzione*	*Azione*
Definizione di una procedura	**FUNCTION... END FUNCTION**	Segnano rispettivamente l'inizio e la fine di una procedura **FUNCTION**.
	SUB...END SUB	Segnano rispettivamente l'inizio e la fine di una procedura **SUB**.
Chiamata di una procedura	**CALL**	Trasferisce il controllo ad una procedura **SUB** del BASIC, o ad una procedura scritta in un altro linguaggio di programmazione e compilata separatamente. (La parola chiave **CALL** è facoltativa.)
Uscita da una procedura	**EXIT FUNCTION**	Fornisce un altro metodo per uscire da una procedura **FUNCTION**.
	EXIT SUB	Fornisce un metodo alternativo per uscire da una procedura **SUB**.
Riferimento ad una procedura prima della sua definizione	**DECLARE**	Dichiara una **FUNCTION** o una **SUB**, determinando facoltativamente il numero ed il tipo dei suoi parametri.
Condivisione di variabili tra moduli, procedure o programmi	**COMMON**	Fa condividere le variabili tra moduli separati. Quando viene utilizzata con l'attributo **SHARED**, essa fa condividere le variabili tra diverse procedure dello stesso modulo. Inoltre, passa i valori delle variabili dal programma corrente a quello nuovo quando il controllo viene trasferito per mezzo dell'istruzione **CHAIN**.

continua

Processo	*Istruzione*	*Azione*
	SHARED	Utilizzata con un'istruzione **COMMON**, **DIM** o **REDIM** a livello di modulo (ad esempio, **DIM SHARED**), fa condividere le variabili tra ogni **SUB** o **FUNCTION** del modulo stesso. Utilizzata da sola all'interno di una procedura, fa condividere le variabili tra quella procedura e il codice a livello di modulo.
Conservazione dei valori di una variabile	**STATIC**	Garantisce che le variabili siano locali rispetto ad una procedura o ad una funzione **DEF FN** e ne conserva i valori tra una chiamata di procedura o di funzione e l'altra.
Definizione di una funzione multiriga	**DEF FN...** **END DEF**	Segnano, rispettivamente, l'inizio e la fine di una funzione **DEF FN** multiriga. (Questo è il vecchio stile di programmazione delle funzioni BASIC – le procedure **FUNCTION** forniscono un'alternativa più evoluta.)
Uscita da una funzione multiriga	**EXIT DEF**	Fornisce un metodo alternativo per uscire da una funzione multiriga **DEF FN**.
Chiamata di una subroutine BASIC	**GOSUB**	Trasferisce il controllo ad una determinata riga di un modulo. Il controllo viene restituito dalla subroutine alla riga successiva all'istruzione **GOSUB**, per mezzo di un'istruzione **RETURN**. (Questo è il vecchio stile di programmazione delle subroutine BASIC – le procedure **SUB** forniscono un'alternativa più evoluta.)
Trasferimento ad un altro programma	**CHAIN**	Trasferisce il controllo dal programma corrente ad un altro; per passare le variabili al nuovo programma utilizzare **COMMON**.

9.3 Riassunto delle istruzioni I/O standard

La tabella 9.3 elenca le istruzioni e le funzioni utilizzate in BASIC per l'I/O standard (generalmente, input dalla tastiera e output sullo schermo).

Tabella 9.3 Istruzioni e funzioni utilizzate per l'I/O standard

Processo	*Istruzione o funzione*	*Azione*
Visualizzazione di testo sullo schermo	**PRINT**	Visualizza del testo sullo schermo. Utilizzando **PRINT** senza argomenti viene visualizzata una riga vuota.
	PRINT USING	Visualizza del testo formattato sullo schermo.
Modifica della lunghezza della riga di output	**WIDTH**	Imposta il numero di colonne visualizzate a 40 o 80 e, sui computer forniti di schede EGA o VGA, controlla il numero di righe visualizzate (25 o 43).
	WIDTH "SCRN:"	Assegna una lunghezza massima alle righe inviate allo schermo, quando utilizzata prima di un'istruzione **OPEN "SCRN:"**.
Lettura dell'input dalla tastiera	**INKEY$**	Legge un carattere dalla tastiera (o una stringa nulla se non è in attesa alcun carattere).
	INPUT$	Legge un determinato numero di caratteri dalla tastiera e li memorizza in un'unica variabile a stringa.
	INPUT	Legge l'input dalla tastiera e lo memorizza in un elenco di variabili.
	LINE INPUT	Legge una riga di input dalla tastiera e la memorizza in un'unica variabile a stringa.

continua

Processo	*Istruzione o funzione*	*Azione*
Posizionamento del cursore sullo schermo	**LOCATE**	Porta il cursore su una determinata riga o colonna.
	SPC	Salta spazi nell'output visualizzato.
	TAB	Visualizza l'output in una data colonna.
Informazioni sulla posizione del cursore	**CSRLIN**	Indica in quale riga si trova il cursore.
	POS (*n*)	Indica la colonna in cui si trova il cursore.
Creazione di una finestra di testo	**VIEW PRINT**	Imposta il margine superiore ed inferiore per la visualizzazione dell'output di testo.

9.4 Riassunto delle istruzioni di I/O su file

La tabella 9.4 elenca le istruzioni e le funzioni utilizzate nella programmazione in BASIC dei file di dati.

Tabella 9.4 Istruzioni e funzioni utilizzate per I/O su file di dati

Processo	*Istruzione o funzione*	*Azione*
Creazione di un nuovo file o accesso ad un file esistente	**OPEN**	Apre un file per la lettura o la memorizzazione di record (I/O).
Chiusura di un file	**CLOSE**	Termina l'I/O su un file.
Scrittura di dati a un file	**PRINT #**	Scrive un elenco di variabili come campi di record in un file aperto in precedenza.*
	PRINT USING #	Analogo a **PRINT #**, tranne per il fatto che **PRINT USING #** formatta i campi dei record.*
	WRITE #	Scrive un elenco di variabili come campi di record in un file aperto in precedenza.*
	WIDTH	Determina la lunghezza standard per i record di un file.*
	PUT	Scrive il contenuto di una variabile di tipo utente in un file aperto in precedenza.†
Lettura di dati da un file	**INPUT #**	Legge i campi da un record ed assegna ciascuno di essi ad una variabile del programma.*
	INPUT$	Legge una stringa di caratteri da un file.
	LINE INPUT #	Legge un record e lo memorizza in un'unica variabile a stringa.*
	GET	Legge i dati da un file e li assegna agli elementi di una variabile di tipo utente.†

continua

Processo	*Istruzione o funzione*	*Azione*
Gestione dei file	**FILES**	Visualizza una lista dei file di una directory specificata.
	FREEFILE	Restituisce il successivo numero di file disponibile.
	KILL	Elimina un file dal disco.
	NAME	Cambia il nome di un file.
Informazioni su un file	**EOF**	Controlla se sono già stati letti tutti i dati di un file.
	FILEATTR	Restituisce il numero assegnato dal sistema operativo ad un file aperto ed un numero che indica la modalità di apertura del file (**INPUT**, **OUTPUT**, **APPEND**, **BINARY**, o **RANDOM**).
	LOC	Fornisce la posizione corrente all'interno di un file. Se il file è ad accesso binario, essa indica la posizione del byte. Se è ad accesso sequenziale, indica la posizione del byte divisa per 128. Se il file è ad accesso casuale, indica il numero dell'ultimo record letto o scritto.
	LOF	Fornisce il numero di byte in un file aperto.
	SEEK (funzione)	Indica il punto dove avrà luogo la successiva operazione I/O. In file ad accesso casuale, essa indica il numero del record successivo da leggere o scrivere. Nelle altre modalità di accesso, indica la posizione del byte successivo da leggere o scrivere.
Spostamenti in un file	**SEEK** (istruzione)	Imposta la posizione del byte per la successiva operazione di lettura o scrittura in un file aperto.

---

* Da utilizzare con file ad accesso sequenziale.

† Da utilizzare con file ad accesso binario o casuale.

9.5 Riassunto delle istruzioni e delle funzioni di manipolazione di stringhe

La tabella 9.5 elenca le istruzioni e le funzioni utilizzate in BASIC per manipolare le stringhe.

Tabella 9.5 Istruzioni e funzioni utilizzate per la manipolazione di stringhe

Processo	*Istruzione o funzione*	*Azione*
Estrazione di parti di una stringa	**LEFT$**	Restituisce un dato numero di caratteri dal lato sinistro di una stringa.
	RIGHT$	Restituisce un dato numero di caratteri dal lato destro di una stringa.
	LTRIM$	Restituisce una copia della stringa priva degli spazi iniziali.
	RTRIM$	Restituisce una copia della stringa priva degli spazi finali.
	MID$ (funzione)	Restituisce un dato numero di caratteri dall'interno di una stringa.
Ricerca di stringhe	**INSTR**	Ricerca una stringa all'interno di un'altra.
Conversione in lettere maiuscole o minuscole	**LCASE$**	Restituisce una copia di una stringa con tutte le lettere maiuscole (A–Z) convertite in minuscole (a–z); lascia inalterate le lettere minuscole e gli altri caratteri.
	UCASE$	Restituisce una copia di una stringa con tutte le lettere minuscole (a–z) convertite in maiuscole (A–Z); lascia inalterate le lettere maiuscole e gli altri caratteri.
Sostituzione di stringhe	**MID$** (istruzione)	Sostituisce parte di una stringa con un'altra stringa.
	LSET	Giustifica a sinistra una stringa all'interno di una stringa a lunghezza fissa.

continua

Processo	*Istruzione o funzione*	*Azione*
	RSET	Giustifica a destra una stringa all'interno di una stringa a lunghezza fissa.
Conversione tra numeri e stringhe	**STR$**	Restituisce la rappresentazione a stringa del valore di un'espressione numerica.
	VAL	Restituisce il valore numerico di un'espressione a stringa.
Conversione di numeri in stringhe per file di dati e viceversa*	**CV***tipo*	Converte i numeri memorizzati come stringhe in numeri in formato Binario Microsoft nei programmi che usano file ad accesso casuale creati con le precedenti versioni del BASIC.
	CV*tipo***MBF**	Converte i numeri memorizzati in formato Binario Microsoft in numeri in formato IEEE.
	MK*tipo***$**	Converte i numeri in formato Binario Microsoft in stringhe memorizzabili in file ad accesso casuale creati con versioni precedenti del BASIC.
	MK*tipo***MBF$**	Converte i numeri in formato IEEE in stringhe in formato Binario Microsoft.
Creazione di stringhe formate da caratteri ripetuti	**SPACE$**	Restituisce una stringa di spazi vuoti di una determinata lunghezza.
	STRING$	Restituisce una stringa formata dalla ripetizione di un dato carattere.
Lettura della lunghezza di una stringa	**LEN**	Indica di quanti caratteri è composta una stringa.
Operazioni con i valori ASCII	**ASC**	Restituisce il valore ASCII del carattere dato.
	CHR$	Restituisce il carattere con il dato valore ASCII.

---

* Ciò non è più necessario per record ad accesso casuale definiti dalle strutture di dati **TYPE...END TYPE**.

9.6 Riassunto delle istruzioni e delle funzioni grafiche

La tabella 9.6 elenca le istruzioni e le funzioni utilizzate in BASIC per la grafica basata sui pixel.

Tabella 9.6 Istruzioni e funzioni utilizzate per l'output grafico

Processo	*Istruzione o funzione*	*Azione*
Impostazione delle caratteristiche dello schermo	**SCREEN**	Specifica una modalità schermo BASIC, la quale determina caratteristiche dello schermo quali la risoluzione e gli intervalli dei numeri dei colori.
Traccia o cancellazione di un solo punto	**PSET**	Dà un determinato colore ad un pixel sullo schermo, utilizzando per predefinizione il colore di primo piano.
	PRESET	Dà un determinato colore ad un pixel sullo schermo, utilizzando per predefinizione il colore di sfondo; l'effetto è di cancellare il pixel.
Disegno di forme	**LINE**	Disegna una retta o un riquadro.
	CIRCLE	Disegna un cerchio o un'ellisse.
	DRAW	Unisce molte caratteristiche di altre istruzioni grafiche BASIC (disegno di linee, spostamento del cursore grafico, proporzionamento di immagini) in un unico macro linguaggio grafico.
Definizione di coordinate di schermo	**VIEW**	Definisce un rettangolo sullo schermo (una finestra) come area per l'output grafico.
	WINDOW	Consente all'utente di reimpostare le coordinate logiche di una finestra o dello schermo.

continua

Processo	*Istruzione o funzione*	*Azione*
	PMAP	Traduce le coordinate fisiche di un pixel nelle coordinate logiche specificate dall'utente per la finestra attuale, o viceversa.
	POINT (*numero*)	Restituisce le coordinate fisiche o logiche correnti del cursore grafico, a seconda del valore di *numero*.
Utilizzo del colore	**COLOR**	Imposta i colori predefiniti utilizzati in output grafici.
	PALETTE	Assegna colori diversi ai numeri dei colori. E' attivo solo su sistemi forniti di una scheda EGA o VGA.
	POINT (*x*, *y*)	Restituisce il numero del colore del pixel le cui coordinate di schermo sono *x* e *y*.
Colorazione di forme chiuse	**PAINT**	Riempie una zona dello schermo con un colore o un motivo.
Animazione	**GET**	Copia una zona rettangolare dello schermo convertendone l'immagine in dati numerici, memorizzandoli in una matrice numerica.
	PUT	Visualizza sullo schermo un'immagine precedentemente copiata da **GET**.
	PCOPY	Copia una pagina di schermo ad un'altra.

9.7 Riassunto delle istruzioni e delle funzioni di rilevamento e gestione

La tabella 9.7 elenca le istruzioni e le funzioni utilizzate dal BASIC per rilevare e gestire errori ed eventi.

Tabella 9.7 Istruzioni e funzioni utilizzate nella gestione degli errori e degli eventi

Processo	*Istruzione o funzione*	*Azione*
Gestione degli errori durante l'esecuzione di un programma	**ON ERROR GOTO** *riga*	Fa saltare il programma alla data *riga*, dove *riga* è un numero o un'etichetta di riga. Il salto avviene ogni volta che si verifica un errore durante l'esecuzione.
	RESUME	Restituisce il controllo al programma dopo aver eseguito una routine di gestione degli errori. Il programma riprende dall'istruzione che ha causato l'errore (**RESUME [0]**), da quella successiva all'istruzione che ha causato l'errore (**RESUME NEXT**), o dalla riga specificata da *riga* (**RESUME** *riga*).
Dati su una condizione di errore	**ERR**	Restituisce il codice relativo a un errore verificatosi durante l'esecuzione.
	ERL	Restituisce il numero della riga nella quale si è verificato l'errore (se nel programma esistono numeri di riga).
	ERDEV	Restituisce un codice di errore particolare all'ultima periferica (ad esempio una stampante) nella quale il DOS ha rilevato un errore.
	ERDV$	Restituisce il nome dell'ultima periferica nella quale il DOS ha rilevato un errore.

continua

Processo	*Istruzione o funzione*	*Azione*
Definizione di codici di errore utente	**ERROR**	Simula la presenza di un errore BASIC; può essere anche utilizzata per definire un errore non rilevato dal BASIC.
Gestione degli eventi durante l'esecuzione di un programma	**ON** *evento* **GOSUB** *riga*	Determina un salto alla subroutine che inizia alla data *riga*, dove *riga* è un numero o etichetta di riga, ogni volta che durante l'esecuzione si verifica l'*evento* dato.
	evento **ON**	Attiva la gestione del dato *evento*.
	evento **OFF**	Disattiva la gestione del dato *evento*.
	evento **STOP**	Sospende la gestione del dato *evento*.
	RETURN	Restituisce il controllo al programma dopo aver eseguito una subroutine per la gestione degli eventi. Il programma riprende dall'istruzione immediatamente successiva al verificarsi dell'evento (**RETURN**), o dalla riga identificata in *riga* (**RETURN** *riga*).

Appendici

A Conversione dei programmi BASICA in QuickBASIC
B Differenze dalle precedenti versioni di QuickBASIC
C Limiti in QuickBASIC
D Codici della tastiera e codici dei caratteri ASCII
E Parole riservate del BASIC
F Metacomandi
G Compilazione ed esecuzione del link da DOS
H Creazione ed utilizzo delle librerie Quick
I Messaggi di errore

Appendice A
Conversione dei programmi BASICA in QuickBASIC

Generalmente, QuickBASIC accetta le istruzioni BASIC scritte per i compilatori del BASIC utilizzati su personal computer IBM e compatibili: l'IBM Advanced Personal Computer BASIC versione A3.00 (BASICA), e il Microsoft GW-BASIC®. Le differenze interne tra QuickBASIC e queste versioni interpretate del BASIC rendono però necessarie alcune modifiche. Poiché tali modifiche si riferiscono ad entrambe le versioni, quest'appendice utilizza il termine BASICA per riferirsi sia al GW-BASIC che al BASICA.

L'appendice fornisce informazioni su:

- Compatibilità del formato dei file sorgenti
- Istruzioni e funzioni non lecite o che richiedono modifiche per essere utilizzate in QuickBASIC
- Differenze tra editor nella gestione delle tabulazioni

Le sezioni seguenti descrivono solo le modifiche necessarie per compilare ed eseguire un programma BASICA con la versione 4.5 di QuickBASIC. I capitoli di questo volume spiegano come utilizzare le caratteristiche di QuickBASIC per potenziare i programmi BASICA già esistenti.

A.1 Il formato del file sorgente

La versione 4.5 di QuickBASIC richiede che il file sorgente sia in formato ASCII o nel formato proprio di QuickBASIC. Un file creato in BASICA deve essere memorizzato con l'opzione ,A; altrimenti il BASICA comprimerà il testo del programma in un formato illeggibile da QuickBASIC. Se ciò si verifica, è opportuno ricaricare il BASICA e memorizzare nuovamente il file in formato ASCII, utilizzando l'opzione ,A. Ad esempio, il file MIOPROG.BAS viene memorizzato in formato ASCII con il seguente comando BASICA:

```
SAVE "MIOPROG.BAS",A
```

A.2 Istruzioni e funzioni non ammesse in QuickBASIC

Le istruzioni e le funzioni sotto elencate non possono essere utilizzate all'interno di un programma QuickBASIC, in quanto esse eseguono operazioni di modifica sul file sorgente, interferiscono con l'esecuzione del programma, si riferiscono ad una periferica a cassetta (non supportata da QuickBASIC) o duplicano il supporto fornito dall'ambiente QuickBASIC:

AUTO	**LIST**	**NEW**
CONT	**LLIST**	**RENUM**
DEF USR	**LOAD**	**SAVE**
DELETE	**MERGE**	**USR**
EDIT	**MOTOR**	

A.3 Istruzioni che richiedono modifiche

Se il programma BASICA contiene qualsiasi istruzione elencata nella tabella A.1, sarà probabilmente necessario modificare il codice sorgente prima di poter eseguire il programma in QuickBASIC.

Tabella A.1 Istruzioni che richiedono modifiche

Istruzione	*Modifica*
BLOAD/BSAVE	Le posizioni di memoria possono essere differenti in QuickBASIC.
CALL	L'argomento è il nome della procedura **SUB** che si sta chiamando.
CHAIN	QuickBASIC non supporta le opzioni **ALL**, **MERGE**, **DELETE** o le opzioni di *numeroriga*.
COMMON	Le istruzioni **COMMON** devono apparire prima di qualunque istruzione eseguibile.
DEF*tipo*	Le istruzioni **DEF***tipo* vanno spostate all'inizio del file sorgente BASICA.
DIM	Tutte le istruzioni **DIM** che dichiarano matrici statiche devono apparire all'inizio dei programmi QuickBASIC.
DRAW, **PLAY**	QuickBASIC richiede che la funzione **VARPTR$** sia utilizzata con variabili incorporate.
RESUME	Se si verifica un errore in una funzione monoriga, QuickBASIC tenta di continuare l'esecuzione del programma alla riga contenente la funzione.
RUN	Nei file eseguibili prodotti da QuickBASIC l'oggetto di un'istruzione **RUN** o **CHAIN** non può essere un file .BAS, ma deve essere un file eseguibile. L'opzione /R del BASICA non viene supportata. All'interno dell'ambiente QuickBASIC, invece, l'oggetto di un'istruzione **RUN** o **CHAIN** è un file .BAS. **RUN** {*numeroriga* | *etichettariga*} è però supportata in QuickBASIC; tale istruzione riavvia il programma dalla riga specificata.

A.4 Differenze tra editor nella gestione delle tabulazioni

Per rappresentare i livelli di indentazione, QuickBASIC utilizza singoli spazi (piuttosto che il carattere di tabulazione letterale ASCII 9). L'opzione **Punti tabulazione** nella finestra di dialogo **Display** del menu **Opzioni** fissa il numero di spazi di ogni livello di indentazione. (Per ulteriori informazioni, consultare il capitolo 20, "Il menu Opzioni", nel volume *L'ambiente di programmazione Microsoft QuickBASIC*.)

Alcuni editor di testo utilizzano il carattere di tabulazione letterale per rappresentare più spazi durante la memorizzazione dei file di testo. L'editor di QuickBASIC considera i caratteri di tabulazione letterale in questi file in questo modo:

- All'interno di stringhe chiuse tra virgolette i caratteri di tabulazione letterali appaiono come il carattere ASCII 9 nella tabella ASCII dell'appendice D, "Codici della tastiera e codici dei caratteri ASCII".
- All'esterno di stringhe racchiuse tra virgolette, i caratteri di tabulazione indentano il testo al successivo livello di indentazione.

Se si tenta di modificare l'impostazione dei **Punti tabulazione** dopo il caricamento di un programma del genere, QuickBASIC visualizza questo messaggio di errore:

```
Impossibile modificare i punti di tabulazione se un file è
residente
```

Ciò serve ad evitare la riformattazione di vecchi file sorgenti creati con altri editor. E' possibile impostare una nuova indentazione per il file con questo procedimento:

1. Memorizzare il file per salvare le modifiche, quindi selezionare il comando **Nuovo programma** dal menu **File**.
2. Selezionare il comando **Display** dal menu **Opzioni** ed impostare l'opzione **Punti tabulazione** come descritto sopra.
3. Selezionare il comando **Apri programma** dal menu **File** e ricaricare il programma. Eseguito ciò, le indentazioni coincideranno con la nuova impostazione dell'opzione **Punti tabulazione**.

Notare che questa procedura si utilizza solo per vecchi programmi. Il testo creato all'interno di QuickBASIC non può essere riformattato in questo modo.

Appendice B
Differenze dalle precedenti versioni di QuickBASIC

La versione 4.5 di QuickBASIC è stata ulteriormente revisionata e perfezionata rispetto alle versioni precedenti. Quest'appendice descrive le differenze tra le versioni 2.0 e 4.5 di QuickBASIC. Salvo indicazione contraria, le differenze tra la versione 2.0 e 4.5. sono valide anche tra le versioni 3.0 e 4.5. Le differenze tra le versioni 4.0 e 4.5 si limitano per lo più all'ambiente QuickBASIC.

Quest'appendice fornisce informazioni sui seguenti perfezionamenti della versione 4.5 di QuickBASIC:

- Capacità e caratteristiche tecniche
- Potenziamenti dell'ambiente
- Potenziamento nella compilazione e nella messa a punto
- Modifiche nel linguaggio
- Compatibilità tra i file

B.1 Caratteristiche di QuickBASIC

Questa sezione confronta le caratteristiche della versione 4.5 di Microsoft QuickBASIC con quelle delle versioni precedenti. La tabella B.1, riportata di seguito, contiene un elenco di queste caratteristiche, ed è seguita da una descrizione più dettagliata.

Tabella B.1 Caratteristiche della versione 4.5 del Microsoft QuickBASIC

	Versione QuickBASIC			
Caratteristica	***2.0***	***3.0***	***4.0***	***4.5***
Consulente QB in linea (guida dettagliata)	No	No	No	Sì
Funzione intercambiabile pulsante destro mouse	No	No	No	Sì
Osservazione immediata di variabili ed espressioni	No	No	No	Sì
Impostazione dei percorsi di ricerca predefiniti	No	No	No	Sì
Tipi di dati definiti dall'utente	No	No	Sì	Sì
Formato IEEE, supporto del coprocessore matematico	No	Sì	Sì	Sì
Guida in linea	No	No	Sì	Sì
Guida ai simboli	No	No	No	Sì
Interi lunghi (a 32 bit)	No	No	Sì	Sì
Stringhe a lunghezza fissa	No	No	Sì	Sì
Verifica della sintassi al momento della digitazione	No	No	Sì	Sì
I/O su file binari	No	No	Sì	Sì
Procedure **FUNCTION**	No	No	Sì	Sì
Supporto del CodeView®	No	No	Sì	Sì
Compatibilità con altri linguaggi	No	No	Sì	Sì
Più moduli in memoria	No	No	Sì	Sì

	Versione QuickBASIC			
Caratteristica	***2.0***	***3.0***	***4.0***	***4.5***
Compatibilità con ProKey, SideKick, SuperKey	No	Sì	Sì	Sì
Modalità inserimento / sovrascrittura	No	Sì	Sì	Sì
Interfaccia tastiera in stile WordStar	No	No	Sì	Sì
Ricursione	No	No	Sì	Sì
Listati di errore durante compilazioni a parte	No	Sì	Sì	Sì
Listati in linguaggio assembler durante compilazioni a parte	No	Sì	Sì	Sì

B.1.1 Caratteristiche nuove nella versione 4.5 di QuickBASIC

Questa nuova versione offre una guida in linea alle parole chiave, ai comandi e ai menu di QuickBASIC, così come ad argomenti generali e alle variabili. Gli esempi visualizzati sulle schermate di guida possono essere copiati ed aggiunti direttamente ai programmi, riducendo così i tempi di sviluppo.

Gli utilizzatori del mouse possono ora impostare la funzione del pulsante destro del mouse con il comando **Pulsante destro mouse** del menu **Opzioni**. Utilizzare la funzione che meglio risponde alle proprie esigenze. Per ulteriori informazioni, consultare il capitolo 19, "Il menu Chiamate", nel manuale *L'ambiente di programmazione Microsoft QuickBASIC*.

Per velocizzare la messa a punto, QuickBASIC ora fornisce il comando **Osservazione immediata**, che permette di identificare immediatamente il valore di una variabile o la condizione (vera o falsa) di un'espressione. Per ulteriori informazioni, consultare il capitolo 18, "Il menu Debug", nel manuale *L'ambiente di programmazione Microsoft QuickBASIC*.

La versione 4.5 permette inoltre di impostare percorsi predefiniti per la ricerca di tipi di file specifici. In questo modo i file possono essere raggruppati in base al tipo in diverse directory. Una volta impostato il nuovo percorso di ricerca, QuickBASIC cercherà sempre nella directory giusta. Si possono impostare percorsi di ricerca predefiniti per file eseguibili, per file da includere, per file di libreria e per file di guida.

B.1.2 Caratteristiche introdotte già nella versione 4.0 di QuickBASIC

Se non si ha familiarità con la versione 4.0 o non si è mai operato con QuickBASIC, sarà necessario riesaminare le seguenti caratteristiche introdotte nella versione 4.0 di QuickBASIC e supportate nella versione attuale.

B.1.2.1 Tipi di dati definiti dall'utente

L'istruzione **TYPE** consente la creazione di tipi di dati composti da elementi semplici. Essi sono simili alle strutture del linguaggio C. I tipi di dati definiti dall'utente sono trattati nel capitolo 3, "I/O su file e su periferica".

B.1.2.2 Formato IEEE e supporto del coprocessore matematico

Il Microsoft QuickBASIC fornisce supporto per i numeri in formato IEEE e per il coprocessore matematico. In mancanza di un coprocessore, QuickBASIC lo emula.

I calcoli eseguiti dai programmi compilati in QuickBASIC sono di solito più precisi e possono produrre risultati diversi da quelli eseguiti nei programmi in BASICA o in versioni QuickBASIC precedenti alla 4.0. I numeri in formato IEEE in precisione semplice hanno una cifra decimale in più. Paragonati ai numeri in precisione doppia in formato Binario Microsoft, quelli in formato IEEE hanno una o due cifre in più nella mantissa ed ampliano l'intervallo dell'esponente.

E' possibile utilizzare le versioni 4.0 e 4.5 QuickBASIC con vecchi programmi e su dati già esistenti nei seguenti modi:

- Utilizzare l'opzione /MBF. In questo modo si potranno utilizzare vecchi programmi e file di dati senza riscrivere i programmi o modificare i file.
- Modificare i file di dati e utilizzare la nuova versione di QuickBASIC per ricompilare i programmi. A lungo andare, questo metodo è preferibile, perché garantisce la compatibilità con versioni successive di QuickBASIC e può generare programmi più veloci. E' necessario modificare solo i file ad accesso casuale contenenti numeri reali in formato binario. I file contenenti solo interi o stringhe di dati possono essere utilizzati senza alcuna modifica. Ulteriori informazioni relative a queste opzioni sono fornite nelle sezioni seguenti.

Nota Se procedure in linguaggio assembler che utilizzano numeri reali vengono chiamate dal programma, devono essere scritte in modo da utilizzare numeri in formato IEEE. Questa è la predefinizione per la versione 5.0 e successive del Macro Assembler Microsoft (MASM). Con le versioni precedenti, assicurarsi di utilizzare l'opzione /R dalla riga di comando o la specifica di assemblaggio 8087.

B.1.2.3 Intervalli dei numeri in formato IEEE

Come mostra l'elenco seguente, i numeri in formato IEEE hanno un intervallo più ampio rispetto a quelli in formato Binario Microsoft:

Tipo di numero	*Intervallo dei valori*
Precisione semplice	da $-3{,}37 * 10^{38}$ a $-8{,}43 * 10^{-37}$ Zero reale da $8{,}43 * 10^{-37}$ a $3{,}37 * 10^{38}$
Doppia precisione	da $-1{,}67 * 10^{308}$ a $-4{,}19 * 10^{-307}$ Zero reale da $4{,}19 * 10^{-307}$ a $1{,}67 * 10^{308}$

I valori in precisione semplice raggiungono una precisione di circa sette cifre, quelli in precisione doppia di 15 o 16 cifre.

Notare che i valori in doppia precisione possono avere un esponente di tre cifre. Ciò può determinare dei problemi con le istruzioni **PRINT USING**.

B.1.2.4 PRINT USING e numeri in formato IEEE

Poiché i numeri in formato IEEE in precisione doppia possono avere esponenti più grandi dei numeri in formato Binario Microsoft, può essere necessario utilizzare un particolare formato esponenziale nelle istruzioni **PRINT USING**. Nei programmi che visualizzano valori con esponenti a tre cifre, è indispensabile il nuovo formato. Per visualizzare i numeri con esponenti di tre cifre, vengono utilizzati cinque elevamenti a potenza (`^^^^^`) invece dei quattro che indicano generalmente il formato esponenziale:

```
PRINT USING "+#.######^^^^^",Circonferenza#
```

Se un esponente è troppo grande rispetto ad un campo, QuickBASIC evidenzia il problema sostituendo alla prima cifra un segno di percentuale (%).

B.1.3 Ricompilazione di vecchi programmi con /MBF

La versione 4.5 QuickBASIC può utilizzare vecchi programmi e vecchi file senza alcuna modifica, se questi vengono ricompilati con l'opzione /MBF dalla riga di comando. Ad esempio, per compilare un programma chiamato `multrgrs.bas`, digitare al prompt del DOS:

```
BC multrgrs /MBF;
```

Eseguire quindi il linkaggio del programma come di consueto. Per ricompilare il programma per mezzo del comando **Crea un file EXE**, è necessario utilizzare l'opzione /MBF all'avviamento di QuickBASIC; poi si può procedere normalmente.

L'opzione /MBF converte i numeri dal formato Binario Microsoft al formato IEEE quando essi vengono letti da un file ad accesso casuale; prima di essere scritti in un file vengono riconvertiti in formato Binario Microsoft. Ciò consente di eseguire calcoli con numeri in formato IEEE mantenendo nei file i numeri in formato Binario Microsoft.

B.1.4 Conversione di file e programmi

Se si decide di convertire i programmi e i file di dati invece di utilizzare l'opzione /MBF, è necessario:

1. Ricompilare i programmi.
2. Convertire i file di dati.

I vecchi programmi di QuickBASIC verranno compilati senza alcuna modifica. Non bisogna però utilizzare l'opzione /MBF durante la ricompilazione, altrimenti il programma non utilizzerà i nuovi file di dati.

E' necessario convertire i file di dati contenenti numeri reali, affinché questi vengano memorizzati in formato IEEE. La versione 4.5 di QuickBASIC è fornita di otto funzioni utili a questo scopo.

Nota Non è necessario convertire i file di dati se essi contengono soltanto dati interi e a stringa. Sono da convertire solo quelli contenenti numeri reali.

La versione 4.5 è fornita delle funzioni **CVS**, **CVD**, **MKS$** e **MKD$**, già familiari, utilizzate per la lettura e la scrittura di numeri reali in file ad accesso casuale. Queste funzioni ora gestiscono i numeri reali memorizzati nei file in formato IEEE e non Binario Microsoft. Per la gestione dei numeri in formato Binario Microsoft, la versione 4.5 fornisce le funzioni **CVSMBF**, **CVDMBF**, **MKSMBF$** e **MKDMBF$**.

Con queste funzioni è possibile scrivere un breve programma che legge record dal vecchio file (utilizzando, se necessario, il formato Binario Microsoft), converte i campi dei numeri reali in formato IEEE, inserisce questi campi in un nuovo record e lo memorizza.

Esempi

Il programma seguente copia su un nuovo file un vecchio file di dati, eseguendo le conversioni necessarie.

```
' Definisce i tipi di buffer del nuovo e vecchio file:
TYPE VecchioBuf
   IdOgg AS STRING * 20
   PosX AS STRING * 4
   PosY AS STRING * 4
   ValFunz AS STRING * 8
END TYPE

TYPE NuovoBuf
   IdOgg AS STRING * 20
   PosX AS SINGLE
   PosY AS SINGLE
   ValFunz AS DOUBLE
END TYPE

' Dichiara i buffer:
DIM VecchioFile As VecchioBuf, NuovoFile AS NuovoBuf

' Apre il nuovo ed il vecchio file di dati:
OPEN "VECMBF.DAT" FOR RANDOM AS #1 LEN = LEN(VecchioFile)
OPEN "NUOVIEEE.DAT" FOR RANDOM AS #2 LEN = LEN(NuovoFile)

I = 1

' Legge il primo dei vecchi record:
GET #1, I, VecchioFile

DO UNTIL EOF(1)

' Sposta i campi ai campi del nuovo record, convertendo
' i campi dei numeri reali:
   NuovoFile.IdOgg = VecchioFile.IdOgg
   NuovoFile.PosX = CVSMBF(VecchioFile.PosX)
   NuovoFile.PosY = CVSMBF(VecchioFile.PosY)
   NuovoFile.ValFunz = CVDMBF(VecchioFile.ValFunz)
```

```
' Scrive i campi convertiti al nuovo file:
   PUT #2, I, NuovoFile
   I = I + 1

' Legge il record successivo dal vecchio file:
   GET #1,I,VecchioFile
LOOP

CLOSE #1, #2
```

Ciascun record dei due file ha quattro campi: un campo di identificazione, due campi contenenti numeri reali in precisione semplice, e un ultimo campo contenente un numero reale in precisione doppia.

La maggior parte del lavoro di conversione viene eseguito per mezzo delle funzioni **CVDMBF** e **CVSMBF**. Ad esempio, la seguente riga di programma converte il campo in doppia precisione:

```
NuovoFile.ValFunz = CVDMBF(VecchioFile.ValFunz)
```

Gli otto byte presenti nel campo del record VecchioFile. ValFunz vengono convertiti da un valore in doppia precisione in formato Binario Microsoft in uno in doppia precisione in formato IEEE mediante la funzione **CVDMBF**. Questo valore viene memorizzato nel campo corrispondente del nuovo record, che poi verrà trascritto nel nuovo file con l'istruzione **PUT**.

B.1.5 Altre caratteristiche di QuickBASIC

La versione 4.0 di QuickBASIC ha introdotto alcune caratteristiche e strumenti che hanno potenziato QuickBASIC in modo da renderlo apprezzato dal professionista senza per questo alienare il principiante. La versione 4.5 supporta queste stesse caratteristiche, insieme a ulteriori potenziamenti.

B.1.5.1 Interi lunghi (a 32 bit)

Gli interi lunghi (a 32 bit) eliminano gli errori di arrotondamento nei calcoli con numeri compresi tra −2 147 483 648 e 2 147 483 647 e permettono di eseguire calcoli molto più velocemente rispetto a quelli in virgola mobile.

B.1.5.2 Stringhe a lunghezza fissa

Le stringhe a lunghezza fissa consentono di incorporare dati di stringa in tipi definiti dall'utente. Per ulteriori informazioni, consultare il capitolo 4, "Manipolazione di stringhe".

B.1.5.3 Verifica della sintassi durante l'immissione

Attivata la verifica della sintassi, QuickBASIC esamina ciascuna riga al momento della digitazione per eventuali errori di sintassi e di doppia definizione. Il capitolo 12, "Utilizzo dell'Editor", nel manuale *L'ambiente di programmazione Microsoft QuickBASIC* tratta la verifica della sintassi e le altre caratteristiche dell'editor intelligente.

B.1.5.4 I/O su file binari

Le versioni 4.0 e 4.5 di QuickBASIC consentono l'accesso ai file binari. Ciò è particolarmente utile per la lettura e la modifica di file memorizzati in formato non ASCII. Il vantaggio maggiore dell'accesso binario è quello di garantire che i file non siano considerati come una raccolta di record. Se un file viene aperto in modalità binaria, è possibile passare a qualunque posizione di byte nel file, e poi leggervi o scrivervi qualunque numero di byte. In questo modo, diverse operazioni di I/O sullo stesso file possono leggere (**GET**) o immettere (**PUT**) un numero diverso di byte – diversamente da quanto accade nei file ad accesso casuale, dove tale numero è stabilito dalla lunghezza predefinita di un solo record.

Per ulteriori informazioni relative all'accesso ai file binari, consultare il capitolo 3, "I/O su file e su periferica".

B.1.5.5 Procedure FUNCTION

Le procedure **FUNCTION** consentono di collocare una funzione in un modulo e di chiamarla da un altro. Per ulteriori informazioni relative all'utilizzo delle procedure **FUNCTION**, consultare il capitolo 2, "Procedure SUB e FUNCTION".

Nelle versioni di QuickBASIC precedenti alla 4.0, una procedura **SUB** e una variabile potevano avere lo stesso nome. Ora, invece, le procedure **SUB** e **FUNCTION** devono avere nomi univoci.

B.1.5.6 Supporto per il debugger CodeView

Gli strumenti BC.EXE e LINK.EXE della riga di comando possono essere utilizzati per creare file eseguibili compatibili con il debugger Microsoft CodeView, un efficace strumento supportato dal Microsoft Macro Assembler (versione 5.0 o successive) e dal Microsoft C (versione 5.0 o successive). I moduli compilati con BC possono essere linkati a quelli compilati in altri linguaggi Microsoft in modo che il risultante file eseguibile possa essere messo a punto con il debugger CodeView. Per ulteriori informazioni, consultare l'appendice G, "Compilazione ed esecuzione del link da DOS".

B.1.5.7 Compatibilità con altri linguaggi

La versione 4.5 di QuickBASIC consente di chiamare routine scritte in altri linguaggi Microsoft utilizzando le convenzioni di chiamata del C o del Pascal. Gli argomenti vengono passati per riferimento di tipo NEAR o FAR, o per valore. Il codice scritto in altri linguaggi può essere inserito in librerie Quick o linkato a file eseguibili.

La versione 4.5 di QuickBASIC non supporta la routine PTR86; bisogna utilizzare, invece, le funzioni **VARPTR** e **VARSEG**.

B.1.5.8 Moduli multipli in memoria

In memoria possono essere caricati più moduli contemporaneamente. Le versioni precedenti alla 4.0 consentivano di caricare in memoria un solo modulo alla volta. Adesso è possibile, invece, modificare, eseguire e mettere a punto programmi a più moduli all'interno dell'ambiente QuickBASIC. Per ulteriori informazioni relative all'utilizzo di moduli multipli, consultare il capitolo 2, "Procedure SUB e FUNCTION", e il capitolo 7, "Programmazione a moduli".

B.1.5.9 Compatibilità con ProKey, SideKick, SuperKey

All'interno dell'ambiente QuickBASIC, si possono utilizzare ProKey, SideKick e SuperKey. Altre riprogrammazioni della tastiera o altri software da tavolo possono non funzionare in QuickBASIC. Rivolgersi ai fornitori o ai produttori di altri programmi per informazioni sulla loro compatibilità o meno con la versione 4.5 di QuickBASIC.

B.1.5.10 Modalità inserimento e sovrascrittura

Premendo il tasto INS, si alterna la modalità inserimento con quella sovrascrittura. In modalità sovrascrittura, il cursore non appare più come una linea, ma come un rettangolino lampeggiante. Notare che il tasto INS ha la stessa funzione della combinazione di tasti CTRL+O nella versione 3.0.

In modalità inserimento, l'editor immette il carattere digitato nel punto in cui è posizionato il cursore. La modalità inserimento è quella predefinita.

B.1.5.11 Comandi dalla tastiera in stile WordStar

QuickBASIC supporta molte delle sequenze di tasti note agli utenti del WordStar. Nel capitolo 12, "Utilizzo dell'editor", del manuale *L'ambiente di programmazione Microsoft QuickBASIC*, è riportato un elenco completo dei comandi stile WordStar.

B.1.5.12 Ricursione

Le versioni 4.0 e 4.5 di QuickBASIC supportano la ricursione, ovvero la capacità di una procedura di autochiamarsi. La ricursione è utile in alcuni procedimenti, quali l'ordinamento. Per ulteriori informazioni relative all'utilizzo della ricursione, consultare il capitolo 2, "Procedure SUB e FUNCTION".

B.1.5.13 Listati di errori durante la compilazione separata

QuickBASIC visualizza messaggi di errore descrittivi durante la compilazione dei programmi per mezzo del comando BC. Utilizzando questo comando, è possibile reindirizzare i messaggi di errore ad un file o ad una periferica per ottenere una copia degli errori di compilazione. Per ulteriori informazioni relative al comando BC, consultare l'appendice G, "Compilazione ed esecuzione del link da DOS".

Esempi

Gli esempi seguenti mostrano come utilizzare la riga del comando BC per far visualizzare gli errori:

Riga di comando	*Azione*
`BC TEST.BAS;`	Compila il file `TEST.BAS` e visualizza gli errori sullo schermo.
`BC TEST.BAS; > TEST.ERR`	Compila il file `TEST.BAS` e reindirizza i messaggi di errore al file `TEST.ERR`.

B.1.5.14 Listati del linguaggio assembler durante la compilazione separata

L'opzione /A del comando BC fornisce un listato del codice in linguaggio assembler generato dal compilatore. Per ulteriori informazioni, si consulti l'appendice G, "Compilazione ed esecuzione del link da DOS".

B.2 Differenze all'interno dell'ambiente

L'ambiente di programmazione QuickBASIC permette ora di selezionare i comandi con maggiore flessibilità; fornisce poi ulteriori opzioni finestra, ed un maggior numero di comandi nei menu. Le sezioni B.2.1–B.2.5 descrivono le differenze nell'ambiente di programmazione tra la versione 4.5 e quelle precedenti.

B.2.1 Scelta di comandi e di opzioni

La versione 4.5 di QuickBASIC rende più facile la selezione dei comandi dai menu e delle opzioni dalle finestre di dialogo.

La versione 4.5 di QuickBASIC consente, inoltre, di selezionare qualunque menu premendo il tasto ALT insieme ad un tasto mnemonico. Ciascun comando dei menu e ciascuna opzione della finestra di dialogo dispone di un proprio tasto mnemonico, che esegue immediatamente il comando o seleziona la voce. I tasti mnemonici appaiono evidenziati.

Nella versione 4.5 il tasto INVIO ha la stessa funziona della BARRA SPAZIATRICE nella versione 3.0 e precedenti. Si può premere il tasto INVIO per eseguire un comando da una finestra di dialogo.

B.2.2 Finestre

La versione 4.5 permette l'utilizzo di massimo due finestre di area di lavoro (note come finestre di visualizzazione), di una finestra di guida e di una finestra di esecuzione immediata. Le versioni precedenti alla 4.0 supportavano soltanto due finestre: una di area di lavoro ed una per i messaggi di errore.

B.2.3 Nuovo menu

La versione 4.5 di QuickBASIC dispone di un nuovo menu, il menu **Opzioni**. Con i suoi comandi è possibile manipolare gli attributi di visualizzazione dello schermo, la funzione del pulsante destro del mouse, il percorso di ricerca predefinito per specifici tipi di file, la verifica della sintassi, e il comando **Menu completi**.

B.2.4 Comandi dei menu

I menu **Debug** e **Guida**, presenti nelle precedenti versioni di QuickBASIC, dispongono ora di alcuni nuovi comandi. La tabella B.2 li elenca insieme a quelli del nuovo menu **Opzioni**.

Tabella B.2 Nuovi comandi nei menu della versione 4.5 del QuickBASIC

Menu	*Comando*	*Descrizione*
Debug	**Osserva**	Aggiunge una variabile o un'espressione alla finestra di osservazione.
	Osservazione immediata (nuovo)	Verifica immediatamente il valore di una variabile o di un'espressione.
	Punto di osservazione	Aggiunge un punto di osservazione alla finestra di osservazione.
	Elimina dall'osservazione	Rimuove selettivamente una voce dalla finestra di osservazione.
	Elimina ogni osservazione	Rimuove tutte le voci dalla finestra di osservazione.
	Analizza il flusso	Attiva e disattiva l'analisi della traccia.
	Resoconto	Registra le ultime 20 istruzioni eseguite.
	Punto di interruzione	Attiva o disattiva un punto di interruzione sulla riga corrente.
	Elimina ogni punto di interruzione	Rimuove tutti i punti di interruzione.
	Interrompi su errore (nuovo)	Arresta l'esecuzione del programma al verificarsi di un errore, a prescindere da qualsiasi gestione degli errori.
	Imposta istruzione successiva	Imposta l'istruzione successiva da eseguire quando un programma sospeso continua l'esecuzione.

continua

Menu	*Comando*	*Descrizione*
Opzione (nuovo)	**Display** (nuovo)	Consente di personalizzare gli elementi dello schermo.
	Percorsi di ricerca (nuovo)	Modifica i percorsi di ricerca predefiniti di specifici tipi di file.
	Pulsante destro mouse (nuovo)	Seleziona la funzione del pulsante destro del mouse.
	Verifica sintassi	Attiva e disattiva il controllo automatico della sintassi.
	Menu completi (nuovo)	Fa alternare tra menu completi e menu brevi.
Guida	**Indice** (nuovo)	Visualizza un elenco in ordine alfabetico delle parole chiave QuickBASIC con una breve descrizione di ciascuna di esse.
	Panoramica (nuovo)	Visualizza un diagramma riassuntivo della guida.
	Argomento	Fornisce informazioni su variabili e parole chiave.
	Uso della Guida (nuovo)	Descrive come usare la Guida e le comuni abbreviazioni delle parole chiave.

Nella versione 3.0, l'opzione **Debug** faceva parte del menu **Esegui**. Nella versione 4.5 si può eseguire la messa a punto in qualunque momento, utilizzando gli appositi comandi del menu **Debug**.

B.2.5 Cambiamenti nei tasti di modifica

L'interfaccia della tastiera del QuickBASIC è stata estesa per comprendere sequenze dei tasti di modifica simili a quelle dell'editor di WordStar. (Per ulteriori informazioni sulla modifica, si consulti il capitolo 13, "Il menu Modifica", nel manuale *L'ambiente di programmazione Microsoft QuickBASIC*.) Le funzioni delle sequenze di tasti della versione 2.0 del QuickBASIC, elencate nella tabella B.3, cambiano nelle versioni 4.0 e 4.5.

Tabella B.3 Cambiamenti nei tasti per la modifica

Funzione	*Tasto QuickBASIC 2.0*	*Tasto QuickBASIC 4.5*
Annulla	MAIUSC+ESC	ALT+BKSP
Taglia	CANC	MAIUSC+CANC
Copia	F2	CTRL+INS
Incolla	INS	MAIUSC+INS
Cancella	--	CANC
Sovrascrivi	--	INS

B.3 Differenze nella compilazione e nella messa a punto

Nell'ambiente di programmazione della versione 4.5 del QuickBASIC, la compilazione e la messa a punto non vengono considerate operazioni separate. Il programma è sempre pronto per l'esecuzione, e durante la programmazione è possibile mettere a punto il proprio codice in diversi modi. Questa sezione presenta le differenze nella compilazione e nella messa a punto tra le versioni 4.0 e 4.5 e quelle precedenti.

B.3.1 Differenze nella riga di comando

Le versioni 4.0 e 4.5 del QuickBASIC supportano solo le opzioni della riga di comando del comando QB, e non quelle del comando BC, della versione 2.0. Per compilare un programma al di fuori dell'ambiente QuickBASIC, si utilizza il comando BC descritto nell'appendice G, "Compilazione ed esecuzione del link da DOS".

Le versioni 4.0 e 4.5 non richiedono alcuna delle opzioni di compilazione elencate nella tabella 4.1 del manuale della versione 2.0 del Microsoft QuickBASIC. Se si cerca di richiamare QuickBASIC utilizzandole come opzioni della riga di comando, apparirà un messaggio di errore. Analogamente, nella versione 3.0 era necessario selezionare alcune opzioni del compilatore dalla finestra di dialogo di compilazione; ciò viene ora eseguito automaticamente. La tabella B.4 descrive il modo in cui ora QuickBASIC supporta la funzionalità di queste opzioni.

Tabella B.4 Opzioni QB e BC non utilizzate nelle versioni 4.0 e 4.5 del QuickBASIC

Versione 2.0	*Versione 4.5*
On Error (/E)	Impostata automaticamente ogni qual volta sia presente un'istruzione **ON ERROR**.
Debug (/D)	Sempre attiva quando si esegue un programma all'interno dell'ambiente QuickBASIC. Nel generare programmi eseguibili su disco o librerie Quick, utilizzare l'opzione **Genera codice Debug**.
Rilevamento continuo (/V) e Rilevamento di eventi (/W)	Impostata automaticamente ogni qual volta sia presente un'istruzione **ON** *evento*.
Resume Next (/X)	Impostata automaticamente ogni qual volta sia presente un'istruzione **RESUME NEXT**.
Matrici ordinate per riga (/R)	Disponibile soltanto durante la compilazione con BC.
Minimizzazione dei dati di stringa (/S)	Opzione predefinita di QB. Per disattivare quest'opzione, si deve compilare dalla riga di comando con BC.

Le opzioni elencate nella tabella B.5 sono ora disponibili per i comandi QB e BC.

Tabella B.5 Opzioni per i comandi QB e BC introdotte nella versione 4.0

Opzione	*Descrizione*
/AH	Consente alle matrici dinamiche di record, di stringhe a lunghezza fissa e di dati numerici di superare i 64K. Se quest'opzione non viene specificata, la dimensione massima di ciascuna matrice è 64K. Notare che quest'opzione non ha alcun effetto sul modo in cui gli elementi dei dati vengono passati alle procedure. (Essa viene utilizzata con i comandi QB e BC.)
/H	Visualizza la più alta risoluzione possibile. Ad esempio, se si possiede una scheda EGA, QuickBASIC visualizza 43 righe e 80 colonne di testo. (Quest'opzione viene utilizzata solo con il comando QB.)
/MBF	Converte i numeri dal formato Binario Microsoft in formato IEEE. Si consulti la sezione B.1.2.3 per maggiori informazioni relative a questa opzione (comandi QB e BC).
/RUN *filesorgente*	Esegue immediatamente il *filesorgente* senza prima visualizzare l'ambiente di programmazione QuickBASIC (comando QB).

B.3.2 Differenze nella compilazione a parte

Le versioni 4.0 e 4.5 non consentono compilazioni a parte con il comando QB. Per compilare e collegare i file senza entrare nell'ambiente di programmazione, si utilizza il comando BC descritto nell'appendice G, "Compilazione ed esecuzione del link da DOS".

B.3.3 Librerie dell'utente e BUILDLIB

Le librerie dell'utente create per versioni precedenti non sono compatibili con le versioni 4.0 e 4.5; è necessario, quindi, ricostruire tali librerie dai file sorgente originali.

Le librerie dell'utente vengono ora chiamate librerie Quick. La loro funzione e il loro utilizzo non hanno subito alcuna modifica. L'estensione del nome dei file di queste librerie ora è .QLB invece di .EXE. Il programma di utilità BUILDLIB non è più necessario, in quanto ora le librerie Quick vengono create dall'interno dell'ambiente di programmazione o dalla riga di comando del link. Per ulteriori informazioni su questo argomento, consultare l'appendice H,"Creazione ed utilizzo delle librerie Quick".

B.3.4 Limitazioni nei file da includere

I file da includere possono contenere le dichiarazioni, ma non le definizioni, di procedure **SUB** o **FUNCTION**. Se è necessario utilizzare un vecchio file da includere contenente tali definizioni, si utilizza il comando **Merge** del menu **File** per inserire nel modulo corrente tale file. Quando si fonde un file da includere contenente una procedura **SUB**, il testo della procedura non appare nella finestra attiva. Per visualizzarlo o modificarlo, si sceglie il comando **SUBroutine** del menu **Visualizza**, quindi si seleziona il nome della procedura nella casella di riepilogo. Dopo aver fuso il testo, è possibile eseguire il programma.

Si può decidere invece di inserire la procedura **SUB** in un modulo separato. In questo caso, bisogna seguire una di queste indicazioni per qualunque variabile condivisa (variabili dichiarate in un'istruzione **COMMON SHARED** o **[RE] DIM SHARED** al di fuori della procedura **SUB**, o in un'istruzione **SHARED** all'interno di questa), perché le variabili così dichiarate sono condivise all'interno di un solo modulo:

- Far condividere le variabili tra i moduli elencandole in istruzioni **COMMON** a livello di modulo in entrambi
- Passare le variabili alla procedura **SUB** in un elenco di argomenti

Per ulteriori informazioni relative ai moduli e alle procedure, consultare il capitolo 2, "Procedure SUB e FUNCTION", ed il capitolo 7, "Programmazione a moduli".

B.3.5 Messa a punto

QuickBASIC ora velocizza le operazioni di messa a punto dei programmi, grazie alle seguenti caratteristiche:

- Più punti di interruzione contemporaneamente
- Espressioni di osservazione, punti di osservazione, ed osservazioni immediate
- Potenziamento dell'analisi del programma (traccia)
- Finestra a tutto schermo che visualizza il testo del programma durante il passo a passo
- La capacità di modificare i valori delle variabili durante l'esecuzione e proseguire
- La capacità di modificare il programma e di continuarne l'esecuzione senza riavviarlo
- L'opzione **Resoconto**
- Il menu **Chiamate**
- La guida ai simboli

Le sequenze dei tasti funzione per la messa a punto elencate nella tabella B.6 sono cambiate rispetto alla versione 2.0:

Tabella B.6 Cambiamenti nei tasti per la messa a punto

Funzione	***Tasto della versione 2.0***	***Tasto della versione 4.5***
Traccia	ALT+F8	F8
Passo	ALT+F9	F10

Notare che l'animazione viene attivata se si attiva il comando **Analizza il flusso** dal menu **Debug** e poi si esegue il programma.

Per ulteriori informazioni, si consulti il capitolo 9, "Messa a punto durante la programmazione", nel manuale *L'ambiente di programmazione Microsoft QuickBASIC*.

B.4 Cambiamenti nel linguaggio BASIC

Questa sezione illustra i potenziamenti e le modifiche del linguaggio BASIC nelle versioni del QuickBASIC 4.5 e precedenti. La tabella B.7 elenca le parole chiave e le relative versioni di QuickBASIC per le quali sono in vigore tali modifiche. La tabella è seguita da una spiegazione più dettagliata di questi cambiamenti.

Tabella B.7 Modifiche introdotte nel linguaggio BASIC

	Versione del QuickBASIC			
Parola chiave	**2.0**	**3.0**	**4.0**	**4.5**
AS	No	No	Sì	Sì
CALL	No	No	Sì	Sì
CASE	No	Sì	Sì	Sì
CLEAR	No	No	Sì	Sì
CLNG	No	No	Sì	Sì
CLS	No	Sì	Sì	Sì
COLOR	No	No	Sì	Sì
CONST	No	Sì	Sì	Sì
CVL	No	No	Sì	Sì
CVSMBF, CVDMBF	No	Sì	Sì	Sì
DECLARE	No	No	Sì	Sì
DEFLNG	No	No	Sì	Sì
DIM	No	No	Sì	Sì
DO...LOOP	No	Sì	Sì	Sì
EXIT	No	Sì	Sì	Sì
FILEATTR	No	No	Sì	Sì
FREEFILE	No	No	Sì	Sì
FUNCTION	No	No	Sì	Sì
GET	No	No	Sì	Sì
LCASE$	No	No	Sì	Sì
LEN	No	No	Sì	Sì
LSET	No	No	Sì	Sì
LTRIM$	No	No	Sì	Sì
MKL$	No	No	Sì	Sì
MKSMBF$, MKDMBF$	No	Sì	Sì	Sì
OPEN	No	No	Sì	Sì
ON UEVENT	No	No	No	Sì
PALETTE	No	No	Sì	Sì

continua

Parola chiave	*Versione del QuickBASIC* 2.0	3.0	4.0	4.5
PUT	No	No	Sì	Sì
RTRIM$	No	No	Sì	Sì
SCREEN	No	No	Sì	Sì
SEEK	No	No	Sì	Sì
SELECT CASE	No	Sì	Sì	Sì
SETMEM	No	No	Sì	Sì
SLEEP	No	No	No	Sì
STATIC	No	No	Sì	Sì
TYPE	No	No	Sì	Sì
UCASE$	No	No	Sì	Sì
UEVENT	No	No	No	Sì
VARPTR	No	No	Sì	Sì
VARSEG	No	No	Sì	Sì
WIDTH	No	Sì	Sì	Sì

La sezione seguente spiega più dettagliatamente le differenze nelle parole chiave sopra elencate.

Parole chiave	*Spiegazione*
AS	La clausola **AS** consente di utilizzare tipi definiti dall'utente nelle istruzioni **DIM**, **COMMON**, e **SHARED**, e negli elenchi dei parametri **DECLARE**, **SUB**, e **FUNCTION**.
CALL	L'utilizzo di **CALL** per chiamare sottoprogrammi dichiarati con l'istruzione **DECLARE** è facoltativo.
CLEAR	L'istruzione **CLEAR** non imposta più la dimensione totale dello stack, ma solo quella richiesta dal programma. QuickBASIC imposta la dimensione dello stack in base alla quantità determinata dall'istruzione **CLEAR** e alle esigenze di QuickBASIC.
CLNG	La funzione **CLNG** arrotonda il proprio argomento e restituisce un intero lungo (a 4 byte) uguale all'argomento.
CLS	L'istruzione **CLS** è stata modificata per cancellare lo schermo con maggiore flessibilità. Notare che QuickBASIC non cancella automaticamente lo schermo all'inizio di ciascun programma, come accadeva nelle versioni precedenti.

Parole chiave	*Spiegazione*
COLOR, SCREEN, PALETTE, WIDTH	Le istruzioni **COLOR, SCREEN, PALETTE**, e **WIDTH** sono ampliate per includere le modalità schermo disponibili con le schede VGA e MCGA dei PS/2 IBM.
CONST	L'istruzione **CONST** consente di definire le costanti simboliche per migliorare la leggibilità del programma e facilitarne la manutenzione.
CVL	La funzione **CVL** viene utilizzata per leggere interi lunghi memorizzati come stringhe nei file di dati ad accesso casuale. **CVL** riconverte una stringa a 4 byte creata con la funzione **MKL$** in un intero lungo da utilizzare nei programmi BASIC.
CVSMBF, CVDMBF	Le funzioni **CVSMBF** e **CVDMBF** convertono le stringhe contenenti numeri in formato Binario Microsoft in numeri in formato IEEE. Sebbene la versione 4.5 del QuickBASIC supporti queste istruzioni, esse sono considerate obsolete poiché il formato IEEE è ora quello standard di QuickBASIC.
DECLARE	L'istruzione **DECLARE** consente di chiamare le procedure da moduli diversi, di verificare il numero ed il tipo degli argomenti passati, e di chiamare le procedure prima che esse siano definite.
DEFLNG	L'istruzione **DEFLNG** dichiara che tutte le variabili, le funzioni **DEF FN**, e le procedure **FUNCTION** sono di tipo ad intero lungo. Vale a dire che una funzione o una variabile è considerata per predefinizione un intero lungo, a meno che non sia stata dichiarata in una clausola **AS** *tipo*, o non abbia un suffisso esplicito di definizione di tipo quale % o $.
DIM	La clausola **TO** dell'istruzione **DIM** consente di specificare indici di qualsiasi valore intero, offrendo una maggiore flessibilità nelle dichiarazioni di matrice.
DO...LOOP	Con l'istruzione **DO...LOOP** si creano dei cicli più potenti che permettono di scrivere i programmi con una struttura migliore.
EXIT	L'utilizzo delle istruzioni **EXIT {DEF | DO | FOR | FUNCTION | SUB}** fornisce comode modalità d'uscita da cicli e da procedure.
FREEFILE, FILEATTR	Le funzioni **FREEFILE** e **FILEATTR** rendono più semplice la scrittura delle applicazioni che eseguono l'I/O su file in un ambiente a più moduli.

continua

Parole chiave	*Spiegazione*
FUNCTION	L'istruzione **FUNCTION...END FUNCTION** permette di definire una procedura multiriga che viene chiamata dall'interno di un'espressione. Queste procedure si comportano in modo molto simile alle funzioni intrinseche quali **ABS** o le funzioni multiriga **DEF FN** nelle versioni 1.0–3.0 del QuickBASIC. A differenza della funzione **DEF FN**, una procedura **FUNCTION** può essere definita in un modulo e chiamata da un altro. Inoltre, le procedure **FUNCTION** supportano le variabili locali e la ricursione.
GET, **PUT**	Per operazioni di I/O, la sintassi delle istruzioni **GET** e **PUT** è stata estesa fino ad includere i record definiti con le istruzioni **TYPE...END TYPE**, e ciò rende inutile l'utilizzo dell'istruzione **FIELD**.
LCASE$**, **UCASE$, **LTRIM$**, **RTRIM$**	Le seguenti funzioni per la gestione di stringhe sono disponibili nella versione 4.5:

Funzione	*Valore di ritorno*
LCASE$	Una copia della stringa con tutti i caratteri minuscoli
UCASE$	Una copia della stringa con tutti i caratteri maiuscoli
LTRIM$	Una copia della stringa priva di tutti gli spazi vuoti iniziali
RTRIM$	Una copia della stringa priva di tutti gli spazi vuoti finali

LEN	L'istruzione **LEN** è stata ampliata perché restituisca il numero di byte richiesto da qualsiasi variabile, costante, espressione o elemento di matrice, sia scalare che di record.
LSET	L'istruzione **LSET** è stata ampliata fino ad includere le variabili di record così come le variabili a stringa. Ciò permette di assegnare una variabile di record ad un'altra variabile di record anche se i record sono di tipo diverso.
MKL$	La funzione **MKL$** viene utilizzata per convertire gli interi lunghi in stringhe memorizzabili in file di dati ad accesso casuale. Utilizzare la funzione **CVL** per riconvertire la stringa in intero lungo.

Parole chiave	*Spiegazione*
MKSMBF$, MKDMBF$	Le funzioni **MKSMBF$** e **MKDMBF$** convertono i numeri in formato IEEE in stringhe contenenti un numero in formato Binario Microsoft. Esse sono superate, anche se ancora supportate nella versione 4.5, che utilizza il formato IEEE.
ON UEVENT	L'istruzione **ON UEVENT** indirizza il programma ad una posizione specificata quando si verifica un evento definito dall'utente (una **UEVENT**). Si utilizzi quest'istruzione analogamente alle altre per la gestione degli eventi.

OPEN	L'istruzione **OPEN** è ora in grado di aprire due file con lo stesso nome per **OUTPUT** e **APPEND** purché i nomi dei percorsi di ricerca siano differenti. Ad esempio, si può avere:
	`OPEN "ESEMPIO" FOR APPEND AS #1`
	`OPEN "TMP\ESEMPIO" FOR APPEND AS #2`
	Alla sintassi dell'istruzione **OPEN**, è stata aggiunta la modalità di accesso ai file binari. Per ulteriori informazioni, si consulti il capitolo 3, "I/O su file e su periferica".

SEEK	L'istruzione e la funzione **SEEK** consentono di portarsi a qualsiasi byte o record di un file. Per maggiori informazioni, consultare il capitolo 3, "I/O su file e su periferica".
SELECT CASE	L'utilizzo delle istruzioni **SELECT CASE** permette di testare una condizione complessa con maggiore semplicità. La clausola **CASE** dell'istruzione **SELECT CASE** accetta ora qualunque espressione (comprese espressioni di variabili) come argomento; nelle versioni precedenti erano consentite soltanto espressioni di costanti.
SETMEM	La funzione **SETMEM** facilita la programmazione in linguaggi misti permettendo di diminuire la quantità di memoria dinamica allocata dal BASIC in modo che possa essere utilizzata da procedure in altri linguaggi.
SLEEP	L'istruzione **SLEEP** fa interrompere il programma per un periodo di tempo determinato, finché l'utente non preme un tasto o sino al verificarsi di un evento gestito. L'argomento opzionale indica in secondi la durata della pausa.

continua

Parole chiave	*Spiegazione*
STATIC	Omettendo l'attributo **STATIC** dalle istruzioni **SUB** e **FUNCTION** le variabili vengono allocate durante la chiamata alla procedura, e non al momento della loro definizione. Tali variabili non conservano i loro valori tra una chiamata alla procedura e l'altra.
TYPE	L'istruzione **TYPE...END TYPE** consente di definire un tipo di dati contenente elementi dei diversi tipi fondamentali. Ciò semplifica la definizione e l'accesso ai record dei file ad accesso casuale.
UEVENT {ON | STOP | OFF}	Attiva, sospende, o disattiva un evento definito dall'utente. Le istruzioni **UEVENT** vengono utilizzate analogamente alle altre istruzioni per la gestione degli eventi.
VARPTR	La funzione **VARPTR** ora restituisce l'offset intero a 16 bit dell'elemento di una variabile o di una matrice BASIC. L'offset parte dall'inizio del segmento contenente la variabile o elemento di matrice.
VARSEG	La funzione **VARSEG** restituisce l'indirizzo del segmento del suo argomento. Ciò permette di impostare appropriatamente **DEF SEG** per utilizzarla con **PEEK**, **POKE**, **BLOAD**, **BSAVE**, e **CALL ABSOLUTE**. Consente anche di ottenere il segmento appropriato da utilizzare con **CALL INTERRUPT** durante l'esecuzione di interrupt del sistema operativo o del BIOS.
WIDTH	Un nuovo argomento dell'istruzione **WIDTH** consente al programma di utilizzare le modalità righe estese su computer forniti di schede EGA, VGA, e MCGA.

Nota Non è più possibile eseguire condizionatamente le istruzioni **NEXT** e **WEND** in un'istruzione monoriga **IF...THEN...ELSE**.

B.5 Compatibilità dei file

I codici sorgenti di tutte le versioni di QuickBASIC sono compatibili tra loro; il codice sorgente creato in una versione precedente può essere compilato nella versione 4.5, tranne che nei casi esposti nella sezione B.3.4, "Limitazioni nei file da includere". La versione 4.5 è in grado di tradurre i file binari della versione 4.0. Se si incontrano difficoltà nella traduzione di altri file binari, memorizzare il programma in formato ASCII (testo) nella versione 4.0, per poi ricaricarlo nella versione 4.5. E' necessario ricompilare i file oggetto e le librerie dell'utente creati con le versioni precedenti di QuickBASIC.

Appendice C
Limiti in QuickBASIC

Sia il compilatore BC che QuickBASIC offrono grande versatilità nella programmazione, ma in ambedue esistono limiti al fine di mantenere ragionevoli le dimensioni e la complessità dei file. Questi limiti possono essere raggiunti in alcune situazioni; essi sono elencati in questa appendice.

Tabella C.1 Limiti in QuickBASIC

Nomi e stringhe	***Massimo***	***Minimo***
Nomi di variabili	40 caratteri	1 carattere
Lunghezza stringa	32767 caratteri	0 caratteri
Interi	32767	−32768
Interi lunghi	2 147 483 647	−2 147 483 648
Numeri in precisione semplice (positivi)	3,402823 E+38	1,401298 E−45
Numeri in precisione semplice (negativi)	−1,401298 E−45	−3,402823 E+38
Numeri in doppia precisione (positivi)	1,797693134862315 D+308	4,940656458412465 D−324
Numeri in doppia precisione (negativi)	−4,940656458412465 D−324	−1,797693134862315 D+38

Matrici	*Massimo*	*Minimo*
Dimensione di una matrice (tutti gli elementi)		
Statica	65535 byte (64K)	1
Dinamica	Memoria disponibile	
Numero di dimensioni di una matrice	8	1
Indici di una matrice	32767	−32768

File e procedure	*Massimo*	*Minimo*
Numero degli argomenti passati ad una procedura	60 interpretati	0
Nidificazione dei file da includere	5 livelli	0
Dimensione di una procedura (interpretata)	65535 byte (64K)	0
Dimensione di un modulo (compilato)	65535 byte (64K)	0
File di dati aperti contemporaneamente	255	0
Numero di record in un file di dati	2 147 483 647	1
Dimensione del record di un file di dati (in byte)	32767 byte (32K)	1 byte
Dimensione di un file di dati	Spazio disponibile sul disco	0
Nomi dei percorsi di ricerca	127 caratteri	1 carattere
Numeri dei messaggi di errore	255	1

Per l'apporto di modifiche nell'ambiente QuickBASIC	*Massimo*	*Minimo*
Caratteri digitati in una casella di testo	128 caratteri	0 caratteri
Stringhe "da trovare"	128 caratteri	1
Stringhe "di sostituzione"	40 caratteri	0
Segnaposto (marker)	4	0
Punti d'osservazione e/o espressioni di osservazione	8	0
Numero di righe nella finestra di esecuzione immediata	10	0
Caratteri nella stessa riga della finestra di visualizzazione	255	0
Lunghezza della stringa **COMMAND$**	124	0

Appendice D
Codici della tastiera e codici dei caratteri ASCII

D.1 Codici della tastiera

Nella tabella della pagina successiva si vedono i codici DOS della tastiera restituiti dalla funzione **INKEY$**.

Le combinazioni di tasti con NUL nella colonna Car restituiscono due byte – un byte nullo (&H00) seguito dal valore riportato nelle colonne Dec e Esa. Ad esempio, premendo ALT+F1 viene restituito un byte nullo seguito da un altro contenente 104 (&H68).

Key	Scan Code		ASCII or Extended†			ASCII or Extended† with Shift			ASCII or Extended† with Ctrl			ASCII or Extended† with Alt		
	Dec	Hex	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
ESC	1	01	27	1B		27	1B		27	1B				
1 !	2	02	49	31	1	33	21	!				120	78	NUL
2 @	3	03	50	32	2	64	40	@	3	03	NUL	121	79	NUL
3 #	4	04	51	33	3	35	23	#				122	7A	NUL
4 $	5	05	52	34	4	36	24	$				123	7B	NUL
5 %	6	06	53	35	5	37	25	%				124	7C	NUL
6 ^	7	07	54	36	6	94	5E	^	30	1E		125	7D	NUL
7 &	8	08	55	37	7	38	26	&				126	7E	NUL
8 *	9	09	56	38	8	42	2A	*				127	7F	NUL
9 (	10	0A	57	39	9	40	28	(				128	80	NUL
0)	11	0B	48	30	0	41	29	)				129	81	NUL
- _	12	0C	45	2D	-	95	5F	_	31	1F		130	82	NUL
= +	13	0D	61	3D	=	43	2B	+				131	83	NUL
BKSP	14	0E	8	08		8	08		127	7F				
TAB	15	0F	9	09		15	0F	NUL						
Q	16	10	113	71	q	81	51	Q	17	11		16	10	NUL
W	17	11	119	77	w	87	57	W	23	17		17	11	NUL
E	18	12	101	65	e	69	45	E	5	05		18	12	NUL
R	19	13	114	72	r	82	52	R	18	12		19	13	NUL
T	20	14	116	74	t	84	54	T	20	14		20	14	NUL
Y	21	15	121	79	y	89	59	Y	25	19		21	15	NUL
U	22	16	117	75	u	85	55	U	21	15		22	16	NUL
I	23	17	105	69	i	73	49	I	9	09		23	17	NUL
O	24	18	111	6F	o	79	4F	O	15	0F		24	18	NUL
P	25	19	112	70	p	80	50	P	16	10		25	19	NUL
[{	26	1A	91	5B	[	123	7B	{	27	1B				
] }	27	1B	93	5D	]	125	7D	}	29	1D				
ENTER	28	1C	13	0D	CR	13	0D	CR	10	0A	LF			
CTRL	29	1D												
A	30	1E	97	61	a	65	41	A	1	01		30	1E	NUL
S	31	1F	115	73	s	83	53	S	19	13		31	1F	NUL
D	32	20	100	64	d	68	44	D	4	04		32	20	NUL
F	33	21	102	66	f	70	46	F	6	06		33	21	NUL
G	34	22	103	67	g	71	47	G	7	07		34	22	NUL
H	35	23	104	68	h	72	48	H	8	08		35	23	NUL
J	36	24	106	6A	j	74	4A	J	10	0A		36	24	NUL
K	37	25	107	6B	k	75	4B	K	11	0B		37	25	NUL
L	38	26	108	6C	l	76	4C	L	12	0C		38	26	NUL
; :	39	27	59	3B	;	58	3A	:						
' "	40	28	39	27	'	34	22	"						
` ~	41	29	96	60	`	126	7E	~						

† I codici estesi restituiscono NUL (ASCII 0) come carattere iniziale. Questo indica che nel buffer delle pressioni di tasti è in attesa un secondo codice (esteso).

Key	Scan Code		ASCII or Extended†			ASCII or Extended† with Shift			ASCII or Extended† with Ctrl			ASCII or Extended† with Alt		
	Dec	Hex	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
L SHIFT	42	2A												
\ |	43	2B	92	5C	\	124	7C	|	28	1C				
Z	44	2C	122	7A	z	90	5A	Z	26	1A		44	2C	NUL
X	45	2D	120	78	x	88	58	X	24	18		45	2D	NUL
C	46	2E	99	63	c	67	43	C	3	03		46	2E	NUL
V	47	2F	118	76	v	86	56	V	22	16		47	2F	NUL
B	48	30	98	62	b	66	42	B	2	02		48	30	NUL
N	49	31	110	6E	n	78	4E	N	14	0E		49	31	NUL
M	50	32	109	6D	m	77	4D	M	13	0D		50	32	NUL
, <	51	33	44	2C	,	60	3C	<						
. >	52	34	46	2E	.	62	3E	>						
/ ?	53	35	47	2F	/	63	3F	?						
R SHIFT	54	36												
* PRTSC	55	37	42	2A	*		INT 5§		16	10				
ALT	56	38												
SPACE	57	39	32	20	SPC	32	20	SPC	32	20	SPC	32	20	SPC
CAPS	58	3A												
F1	59	3B	59	3B	NUL	84	54	NUL	94	5E	NUL	104	68	NUL
F2	60	3C	60	3C	NUL	85	55	NUL	95	5F	NUL	105	69	NUL
F3	61	3D	61	3D	NUL	86	56	NUL	96	60	NUL	106	6A	NUL
F4	62	3E	62	3E	NUL	87	57	NUL	97	61	NUL	107	6B	NUL
F5	63	3F	63	3F	NUL	88	58	NUL	98	62	NUL	108	6C	NUL
F6	64	40	64	40	NUL	89	59	NUL	99	63	NUL	109	6D	NUL
F7	65	41	65	41	NUL	90	5A	NUL	100	64	NUL	110	6E	NUL
F8	66	42	66	46	NUL	91	5B	NUL	101	65	NUL	111	6F	NUL
F9	67	43	67	43	NUL	92	5C	NUL	102	66	NUL	112	70	NUL
F10	68	44	68	44	NUL	93	5D	NUL	103	67	NUL	113	71	NUL
NUM	69	45												
SCROLL	70	46												
HOME	71	47	71	47	NUL	55	37	7	119	77	NUL			
UP	72	48	72	48	NUL	56	38	8						
PGUP	73	49	73	49	NUL	57	39	9	132	84	NUL			
GREY -	74	4A	45	2D	-	45	2D	-						
LEFT	75	4B	75	4B	NUL	52	34	4	115	73	NUL			
CENTER	76	4C				53	35	5						
RIGHT	77	4D	77	4D	NUL	54	36	6	116	74	NUL			
GREY +	78	4E	43	2B	+	43	2B	+						
END	79	4F	79	4F	NUL	49	31	1	117	75	NUL			
DOWN	80	50	80	50	NUL	50	32	2						
PGDN	81	51	81	51	NUL	51	33	3	118	76	NUL			
INS	82	52	82	52	NUL	48	30	0						
DEL	83	53	83	53	NUL	46	2E	.						

† I codici estesi restituiscono NUL (ASCII 0) come carattere iniziale. Questo indica che nel buffer delle pressioni di tasti è in attesa un secondo codice (esteso).

§ Nel DOS, la combinazione dei tasti MAIUSC+STAMP produce un interrupt 5, che stampa lo schermo, a meno che non sia stato definito un gestore di interrupt diverso da quello predefinito.

D.2 Codici dei caratteri ASCII

Ctrl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01	☺	SOH
^B	2	02	☻	STX
^C	3	03	♥	ETX
^D	4	04	♦	EOT
^E	5	05	♣	ENQ
^F	6	06	♠	ACK
^G	7	07	•	BEL
^H	8	08	◘	BS
^I	9	09	○	HT
^J	10	0A	◙	LF
^K	11	0B	♂	VT
^L	12	0C	♀	FF
^M	13	0D	♪	CR
^N	14	0E	♫	SO
^O	15	0F	☼	SI
^P	16	10	►	DLE
^Q	17	11	◄	DC1
^R	18	12	↕	DC2
^S	19	13	‼	DC3
^T	20	14	¶	DC4
^U	21	15	§	NAK
^V	22	16	▬	SYN
^W	23	17	↨	ETB
^X	24	18	↑	CAN
^Y	25	19	↓	EM
^Z	26	1A	→	SUB
^[	27	1B	←	ESC
^\	28	1C	∟	FS
^]	29	1D	↔	GS
^^	30	1E	▲	RS
^_	31	1F	▼	US

Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	32	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	¦
125	7D	}
126	7E	~
127	7F	Δ†

† Il codice ASCII 127 corrisponde al codice del tasto CANC. Nel DOS, questo codice produce lo stesso effetto del codice ASCII 8 (BS). Il codice CANC può essere generato dalla combinazione dei tasti CTRL+BKSP.

Dec	Hex	Char
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	ö
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	₧
159	9F	ƒ

Dec	Hex	Char
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	░
177	B1	▒
178	B2	▓
179	B3	│
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐

Dec	Hex	Char
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	─
197	C5	┼
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	═
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	█
220	DC	▄
221	DD	▌
222	DE	▐
223	DF	▀

Dec	Hex	Char
224	E0	α
225	E1	β
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	µ
232	E7	τ
232	E8	Φ
233	E9	Θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	φ
238	EE	ε
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
245	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	∙
250	FA	·
251	FB	√
252	FC	ⁿ
253	FD	²
254	FE	■
255	FF	

D.3 Traduzione dei nomi dei tasti

Inglese	*Italiano*
Alt	ALT
Alt Gr	ALT GR
Backspace	BACKSPACE
Break	INTERR
Caps Lock	BLOC MAIUS
Ctrl	CTRL
Del	CANC
Down arrow key	GIU'
End	FINE
Enter	INVIO
Esc	ESC
F1...F10	F1...F10
Home	HOME
Ins	INS
Left arrow key	SINISTRA
Num Lock	BLOC NUM
Pause	PAUSA
PgDn	PGGIU'
PgUp	PGSU
Print Screen	STAMP
Right arrow key	DESTRA
Scroll Lock	BLOC SCORR
Shift	MAIUSC
Space bar	BARRA SPAZIATRICE
Sys Req	RSIST
Tab	TAB
Up arrow key	SU

Appendice E
Parole riservate del BASIC

L'elenco seguente contiene le parole riservate del Microsoft BASIC:

ABS
ACCESS
ALIAS
AND
ANY
APPEND
AS
ASC
ATN
BASE
BEEP
BINARY
BLOAD
BSAVE
BYVAL
CALL
CALLS
CASE
CDBL
CDECL
CHAIN
CHDIR
CHR$
CINT
CIRCLE
CLEAR
CLNG
CLOSE
CLS
COLOR
COM
COMMAND$
COMMON
CONST
COS
CSNG
CSRLIN
CVD
CVDMBF
CVI
CVL
CVS
CVSMBF
DATA
DATE$
DECLARE
DEF
DEFDBL
DEFINT
DEFLNG
DEFSNG
DEFSTR
DIM
DO
DOUBLE
DRAW
ELSE
ELSEIF
END
ENDIF
ENVIRON
ENVIRON$
EOF
EQV
ERASE
ERDEV
ERDEV$
ERL
ERR
ERROR
EXIT
EXP
FIELD
FILEATTR
FILES
FIX
FOR
FRE
FREEFILE
FUNCTION
GET
GOSUB
GOTO
HEX$
IF
IMP
INKEY$
INP

INPUT
INPUT$
INSTR
INT
INTEGER
IOCTL
IOCTL$
IS
KEY
KILL
LBOUND
LCASE$
LEFT$
LEN
LET
LINE
LIST
LOC
LOCAL
LOCATE
LOCK
LOF
LOG
LONG
LOOP
LPOS
LPRINT
LSET
LTRIM$
MID$
MKD$
MKDIR
MKDMBF$
MKI$
MKL$
MKS$
MKSMBF$
MOD
NAME
NEXT
NOT
OCT$
OFF
ON
OPEN
OPTION
OR
OUT
OUTPUT
PAINT
PALETTE
PCOPY
PEEK
PEN
PLAY
PMAP
POINT
POKE
POS
PRESET
PRINT
PSET
PUT
RANDOM
RANDOMIZE
READ
REDIM
REM
RESET
RESTORE
RESUME
RETURN
RIGHT$
RMDIR
RND
RSET
RTRIM$
RUN
SADD
SCREEN
SEEK
SEG
SELECT
SETMEM
SGN
SHARED
SHELL
SIGNAL
SIN
SINGLE
SLEEP
SOUND
SPACE$
SPC
SQR
STATIC
STEP
STICK
STOP
STR$
STRIG
STRING
STRING$
SUB
SWAP
SYSTEM
TAB
TAN
THEN
TIME$
TIMER
TO
TROFF
TRON
TYPE
UBOUND
UCASE$
UEVENT
UNLOCK
UNTIL
USING
VAL
VARPTR
VARPTR$
VARSEG
VIEW
WAIT
WEND
WHILE
WIDTH
WINDOW
WRITE
XOR

Appendice F
Metacomandi

In questa appendice vengono descritti i metacomandi di QuickBASIC, comandi che indicano a QuickBASIC di manipolare il programma in un modo particolare. La prima sezione descrive il formato utilizzato per i metacomandi; le due sezioni successive descrivono i metacomandi in dettaglio.

Per mezzo dei metacomandi è possibile:

- Aggiungere alla memoria e compilare altri file sorgenti BASIC, includendoli in punti specifici durante la compilazione (**$INCLUDE**)
- Controllare l'allocazione delle matrici dimensionate (**$STATIC** e **$DYNAMIC**)

F.1 Sintassi dei metacomandi

I metacomandi iniziano con un segno di dollaro ($) e sono sempre racchiusi in un commento di programma. In un solo commento possono essere inclusi più metacomandi, separati tra loro da caratteri di spaziatura: spazio o tabulazione. Due punti separano un metacomando dal relativo argomento:

REM $ *metacomando* [**:***argomento*]

Gli argomenti a stringa devono essere racchiusi tra singole virgolette. I caratteri di spaziatura tra gli elementi di un metacomando vengono ignorati. Le forme seguenti di metacomando sono tutte valide:

```
REM $STATIC $INCLUDE:  'defdati.bi'
REM   $STATIC   $INCLUDE:   'defdati.bi'
' $STATIC $INCLUDE:  'defdati.bi'
'   $STATIC   $INCLUDE:   'defdati.bi'
```

Notare la mancanza di spazi tra il segno del dollaro ed il resto del metacomando.

Se in una descrizione si vuole indicare un metacomando senza farlo eseguire, far precedere il primo segno di dollaro nella riga da un carattere che non sia una tabulazione o uno spazio. Ad esempio, entrambi i metacomandi della riga successiva verranno ignorati:

```
REM x$STATIC $INCLUDE:  'defdati.bi'
```

F.2 Elaborazione di file sorgente aggiuntivi: $INCLUDE

Il metacomando **$INCLUDE** indica al compilatore di sospendere temporaneamente l'elaborazione del file e di leggere piuttosto le istruzioni del programma nel file BASIC specificato dall'argomento. Quando il compilatore raggiunge la fine del file da includere, torna ad elaborare il file originale. Poiché la compilazione inizia dalla riga immediatamente successiva a quella contenente il metacomando **$INCLUDE**, questo deve essere l'ultima istruzione della riga. L'istruzione seguente è corretta:

```
DEFINT I-N  '$INCLUDE: 'COMMON.BAS'
```

Esistono due limitazioni nell'utilizzo dei file da includere:

- I file da includere non devono contenere istruzioni **SUB** o **FUNCTION**.
- I file da includere creati in BASICA devono essere stati memorizzati con l'opzione ,A.

F.3 Allocazione di matrici dimensionate: $STATIC e $DYNAMIC

I metacomandi **$STATIC** e **$DYNAMIC** indicano al compilatore come allocare memoria per le matrici. Nessuno di questi metacomandi assume un argomento:

```
' Rende dinamiche tutte le matrici:
'$DYNAMIC
```

$STATIC riserva lo spazio memoria per le matrici durante la compilazione. In presenza del metacomando **$STATIC**, l'istruzione **ERASE** reinizializza i valori di tutte le matrici a zero (matrici numeriche) o alla stringa nulla (matrici di stringhe), ma non rimuove la matrice. L'istruzione **REDIM** non ha alcun effetto sulle matrici **$STATIC**.

$DYNAMIC riserva lo spazio memoria per le matrici durante l'esecuzione del programma. Ciò significa che un'istruzione **ERASE** rimuoverà la matrice e ne libererà per altro uso la memoria utilizzata. E' possibile utilizzare l'istruzione **REDIM** per modificare la dimensione di una matrice **$DYNAMIC**.

I metacomandi **$STATIC** e **$DYNAMIC** hanno effetto su tutte le matrici tranne che su quelle dimensionate implicitamente (matrici non dichiarate con un'istruzione **DIM**). Le matrici dimensionate implicitamente vengono sempre allocate come se fosse stato utilizzato il metacomando **$STATIC**.

Appendice G
Compilazione ed esecuzione del link da DOS

Quest'appendice spiega come compilare ed eseguire il link al di fuori dell'ambiente QuickBASIC. Ciò può risultare preferibile per una delle seguenti ragioni:

- Per utilizzare un diverso editor di testo
- Per creare programmi eseguibili che possano essere messi a punto con il debugger Microsoft CodeView
- Per creare file di lista da utilizzare nella messa a punto di un programma eseguibile autonomo
- Per utilizzare opzioni non disponibili all'interno dell'ambiente QuickBASIC, quali la memorizzazione di matrici in ordine di riga
- Per eseguire un link con NOEM.OBJ (file fornito col QuickBASIC), il quale riduce la dimensione dei file eseguibili in programmi che vengono sempre utilizzati in presenza di un coprocessore matematico

A lettura ultimata l'utente sarà in grado di:

- Compilare dalla riga di comando del DOS con il comando BC
- Creare file eseguibili e sottoporre al linker dei file programma oggetto con il comando LINK
- Creare e aggiornare librerie autonome (.LIB) con il comando LIB

G.1 BC, LINK e LIB

Nella confezione Microsoft QuickBASIC vengono forniti anche i programmi BC, LINK, e LIB. L'elenco seguente descrive l'utilizzo di questi strumenti per la compilazione e l'esecuzione del link al di fuori dell'ambiente QuickBASIC:

Programma	*Funzione*
BC.EXE	Quando si sceglie il comando **Crea un file EXE** o **Crea libreria** dal menu **Esegui**, QuickBASIC richiama il compilatore BC per creare dei file programma intermedi chiamati file oggetto. Questi poi verranno collegati tra loro per formare il programma o la libreria Quick. BC si utilizza anche qualora si volessero compilare programmi al di fuori dell'ambiente QuickBASIC. C'è chi preferisce compilare con BC i programmi scritti con un altro editor di testo; i due soli casi, però, in cui è necessario utilizzare BC dalla riga di comando sono se il programma è troppo esteso per essere compilato in memoria dallo ambiente QuickBASIC, oppure se si desidera che i file eseguibili siano compatibili con il debugger Microsoft CodeView.
LINK.EXE	Per creare un file eseguibile, QuickBASIC collega i file oggetto creati da BC alle librerie appropriate con il Microsoft Overlay Linker (LINK). LINK può essere utilizzato direttamente se si vogliono linkare file oggetto o creare librerie Quick.
LIB.EXE	Il Microsoft Library Manager (LIB) crea librerie autonome con i file oggetto prodotti da BC. QuickBASIC stesso utilizza LIB per creare tali librerie e si serve poi di esse quando l'utente sceglie il comando **Crea un file EXE** dal menu **Esegui**.

G.2 Il processo di compilazione e di collegamento (linkaggio)

Per creare un programma autonomo da un file sorgente BASIC al di fuori dell'ambiente QuickBASIC, si segua questa procedura:

1. Compilare ciascun file sorgente, creando file oggetto.
2. Collegare i file oggetto utilizzando LINK. Questo comando include una o più librerie autonome e crea un file eseguibile. Prima di creare un file eseguibile, LINK si assicura che tutte le chiamate di procedura nei file sorgenti corrispondano a procedure nelle librerie o presenti in altri file oggetto.

La compilazione ed il linkaggio possono essere effettuati in uno dei seguenti modi:

- Compilare ed eseguire il link in due fasi distinte utilizzando i comandi BC e LINK.
- Creare un file batch contenente tutti i comandi per la compilazione ed il linkaggio. Questo metodo risulta più efficace se si utilizzano le stesse opzioni ogni volta che si compilano e collegano dei programmi.

Nota Quando QuickBASIC compila e collega un programma dall'interno dell'ambiente, l'opzione /E del linker viene impostata automaticamente. Quando però si utilizza il comando LINK al di fuori dell'ambiente QuickBASIC, è necessario specificare esplicitamente l'opzione /E per ridurre al minimo la dimensione del file ed aumentare la velocità di caricamento del programma.

Durante la compilazione e l'esecuzione del link da DOS, non vengono utilizzati i percorsi definiti nel menu **Opzioni**. Per far effettuare la ricerca dei file da includere e dei file libreria secondo le specifiche nel menu **Opzioni**, bisogna impostare le variabili di ambiente LIB e INCLUDE del DOS perché indichino le directory appropriate. Altrimenti, il compilatore e/o il linker potrebbero generare errori del tipo: `File non trovato`.

Le sezioni G.3 e G.4 spiegano come compilare ed eseguire il link in due fasi distinte.

G.3 Compilazione con il comando BC

La compilazione può essere effettuata con il comando BC in uno dei seguenti modi:

- Digitare tutte le informazioni su un'unica riga di comando, utilizzando la sintassi seguente:

  BC *filesorgente* [, [*fileoggetto*] [, [*filelista*]]] [*elencopzioni*] [;]
- Digitare

  `bc`

  e rispondere alle seguenti richieste:

  ```
  Nome del file sorgente [.BAS]:
  Nome del file oggetto [nomebase.OBJ]:
  Listato sorgente [NUL.LST]:
  ```

La tabella G.1 illustra l'input da digitare sulla riga del comando BC o in risposta a ciascun prompt.

Tabella G.1 Input al comando BC

Campo	Prompt	Input
filesorgente	"Nome del file sorgente"	Il nome del file sorgente.
fileoggetto	"Nome del file oggetto"	Il nome del file oggetto che si sta creando.
filelista	"Listato sorgente"	Il nome del file contenente un elenco del sorgente. Il file di lista sorgente contiene l'indirizzo di ciascuna riga del file sorgente, il suo testo e la sua dimensione, e qualsiasi messaggio di errore generato durante la compilazione.
elencopzioni	Fornisce le opzioni dopo una risposta	Qualunque opzione del compilatore descritta nella sezione G.3.2, "Utilizzo delle opzioni del comando BC".

G.3.1 Specificazione dei nomi dei file

Il comando BC opera con alcuni presupposti sui nomi di percorso di ricerca e sulle estensioni dei nomi dei file specificati. Le sezioni successive spiegano meglio questi presupposti e le altre regole utilizzate per specificare i nomi dei file al comando BC.

G.3.1.1 Lettere maiuscole e minuscole

Si può utilizzare qualunque combinazione di lettere maiuscole e minuscole non accentate per i nomi dei file, perchè il compilatore accetta entrambe indistintamente.

Esempio

Il comando BC considera uguali i seguenti nomi di file:

```
abcde.bas
ABCDE.BAS
aBcDe.Bas
```

G.3.1.2 Estensioni dei nomi dei file

Il nome di un file DOS è composto da due parti: il "nome di base", che include tutti i caratteri prima del punto (.), e la "estensione", che comprende il punto e i tre caratteri successivi. Generalmente, l'estensione identifica il tipo del file (file sorgente BASIC, file oggetto, file eseguibile, o libreria autonoma).

I comandi BC e LINK utilizzano le estensioni dei nomi dei file descritte nell'elenco seguente:

Estensione	*Descrizione del file*
.BAS	File sorgente BASIC
.OBJ	File oggetto
.LIB	File libreria autonomo
.LST	File di lista generato da BC
.MAP	File di simboli del programma sottoposto al linker
.EXE	File eseguibile

G.3.1.3 Percorsi di ricerca

Qualunque nome di file può includere il nome completo o parziale del percorso di ricerca. Un nome completo inizia con il nome dell'unità disco; uno parziale consiste in uno o più nomi di directory che precedono il nome del file, ma non include il nome dell'unità disco.

Fornendo i nomi completi dei percorsi di ricerca si possono specificare file con diversi percorsi di ricerca come input per il comando BC, o creare file su diverse unità disco o in differenti directory sull'unità corrente.

Nota Per file che si stanno creando con BC, è possibile specificare percorsi terminanti con una barra rovesciata (\). Nel creare i file, il comando BC utilizza i nomi di file predefiniti.

G.3.2 Utilizzo delle opzioni del comando BC

Le opzioni relative al comando BC sono composte da una sbarra (/) o un trattino (–) seguiti da una o più lettere. (La sbarra ed il trattino possono essere utilizzati indistintamente. In questo manuale vengono utilizzate le sbarre.)

Le opzioni della riga del comando BC sono descritte qui di seguito.

Opzione	*Descrizione*
/A	Crea un listato del codice oggetto disassemblato per ciascuna riga sorgente e mostra il codice in linguaggio assembler generato dal compilatore.
/AH	Consente alle matrici dinamiche di record, di stringhe a lunghezza fissa e di dati numerici di utilizzare tutta la memoria disponibile. Se questa opzione non viene specificata, la dimensione massima di ciascuna matrice è 64K. Notare che questa opzione non ha alcun effetto sul modo in cui gli elementi dei dati vengono passati alle procedure.
/C:*dimensione-buffer*	Imposta la dimensione del buffer per la ricezione di dati a distanza per ciascuna porta di comunicazione, quando si utilizza una scheda per comunicazioni asincrone. (Al buffer di trasmissione sono assegnati 128 byte per ciascuna porta di comunicazione e questo valore non può essere modificato dalla riga del comando BC.) Quest'opzione non produce alcun effetto in assenza di scheda per comunicazioni asincrone. La dimensione totale del buffer per entrambe le porte è, per predefinizione, di 512 byte; la dimensione massima consentita è di 32767 byte.
/D	Genera il codice di messa a punto per la verifica degli errori durante l'esecuzione ed attiva la funzione della combinazione dei tasti CTRL+INTERR. Questa opzione è uguale all'opzione **Genera codice Debug** del comando **Crea un file EXE** del menu **Esegui** all'interno dell'ambiente QuickBASIC.
/E	Indica la presenza di istruzioni **ON ERROR** e **RESUME** *numeroriga* (consultare anche l'opzione /X).
/MBF	Le funzioni intrinseche **MKS$**, **MKD$**, **CVS** e **CVD** vengono convertite rispettivamente in **MKSMBF**, **MKDMBF$**, **CVSMBF** e **CVDMBF**. Ciò consente al programma di leggere e scrivere valori in virgola mobile memorizzati in formato Binario Microsoft.
/O	Sostituisce la libreria di esecuzione BCOM45.LIB con la libreria BRUN45.LIB. Per ulteriori informazioni relative all'utilizzo di queste librerie, consultare il capitolo 16, "Il menu Esegui", nel manuale *L'ambiente di programmazione Microsoft QuickBASIC*.
/R	Memorizza le matrici in ordine per riga. Il BASIC memorizza normalmente le matrici in ordine per colonna. Questa opzione è conveniente se si stanno utilizzando routine di altri linguaggi che memorizzano le matrici in ordine per riga.
/S	Scrive le stringhe racchiuse tra virgolette nel file oggetto invece che nella tabella simbolica. Utilizzare quest'opzione se in un programma con molte costanti a stringa viene visualizzato il messaggio di errore `Memoria esaurita`.

Opzione	*Descrizione*
/V	Attiva la gestione degli eventi di comunicazione (**COM**), della penna ottica (**PEN**), del joystick (**STRIG**), del temporizzatore (**TIMER**), del buffer di suono (**PLAY**) e dei tasti funzione (**KEY**). Utilizzare quest'opzione per controllare il verificarsi di un evento tra un'istruzione e l'altra.
/W	Attiva la gestione degli stessi eventi di /V, ma controlla il verificarsi di un evento ad ogni numero o etichetta di riga.
/X	Indica la presenza dell'istruzione **ON ERROR** insieme ad un'istruzione **RESUME**, **RESUME NEXT** o **RESUME 0**.
/ZD	Genera un file oggetto contenente record dei numeri di riga corrispondenti ai numeri di riga del file sorgente. Quest'opzione è utile per eseguire la messa a punto al livello del codice sorgente utilizzando il Microsoft Symbolic Debug (SYMDEB), disponibile con il Microsoft Macro Assembler versione 4.0.
/ZI	Produce un file oggetto con informazioni per la messa a punto utilizzate dal debugger Microsoft CodeView, disponibile con il linguaggio Microsoft C versione 5.0 e successive, e con il Microsoft Macro Assembler versione 5.0 e successive.

G.4 Esecuzione del link

Dopo la compilazione, è necessario collegare il file oggetto alle librerie appropriate per creare un programma eseguibile. Il comando LINK si può utilizzare in uno dei seguenti modi:

- Fornendo l'input sulla riga di comando secondo questo formato:

  LINK *fileoggetto* [, [*fileseguibile*] [, [*filemap*] [, [*lib*]]] [*opzlink*] [;]]

  La riga di comando non può contenere più di 128 caratteri.

- Digitando

  ```
  link
  ```

  e rispondendo alle seguenti richieste:

  ```
  Moduli oggetto [.OBJ]:
  File eseguibile [nomebase.EXE]:
  File di lista [NUL.MAP]:
  Librerie [.LIB]:
  ```

continua

Per fornire più file ad un prompt, digitare un segno più (+) alla fine della riga. Il prompt riapparirà sulla riga successiva, dando così la possibilità di digitare ulteriore input al prompt.

- Creando un "file risposte" contenente le risposte ai prompt del comando LINK e digitando poi un comando LINK in questa forma:

  LINK @*nomefile*

  nomefile rappresenta qui il nome del file risposte. Delle opzioni per il linker possono essere aggiunte a qualsiasi risposta o possono comparire su una o più righe distinte. Le risposte devono rispettare l'ordine dei prompt del comando LINK esaminati qui sopra.

  Si può digitare il nome di un file risposte dopo qualsiasi prompt del linker, o in qualsiasi punto della riga di comando LINK.

Nella tabella G.2 si vede l'input da digitare sulla riga di comando LINK o in risposta a ciascun prompt.

Tabella G.2 Input al comando LINK

Campo	***Prompt***	***Input***
fileogg	"Moduli oggetto"	Uno o più file oggetto su cui si sta eseguendo il link. Devono essere separati da segni più o da spazi.
filexe	"File eseguibile"	Nome del file eseguibile che si sta creando, qualora gli si voglia assegnare un nome o un'estensione diversa da quella predefinita. E' opportuno utilizzare sempre l'estensione .EXE, poiché il DOS la presuppone nei file eseguibili.
filemap	"File di lista"	Nome del file contenente un listato dei simboli, se se ne sta creando uno.* Si può inoltre specificare uno dei seguenti nomi di periferica DOS per indirizzare il file di lista a quella periferica: AUX per una periferica ausiliaria, CON per la console (terminale), PRN per una stampante, o NULL per nessuna periferica (per non creare alcun file di lista). La sezione G.4.6.11 include un file di lista di esempio e informazioni sul suo contenuto.

Campo	*Prompt*	*Input*
lib	"Librerie"	Una o più librerie autonome (o directory da ricercare per librerie autonome) separate da segni più o da spazi. Il prompt "Librerie" permette di specificare fino a 16 librerie. Qualunque libreria in più sarà ignorata. Nella sezione G.4.3 sono indicate le regole per specificare al linker i nomi delle librerie.
opzlink	Fornisce le opzioni relative a ciascuna risposta	Qualunque opzione LINK descritta nelle sezioni G.4.6.2–G.4.6.15. Le opzioni di LINK possono essere specificate in qualsiasi punto della riga di comando.

* Un altro modo per creare un file di lista map consiste nello specificare l'opzione /MAP al comando LINK (sezione G.4.6.11).

Qualora si stia utilizzando un file risposte, ciascuna risposta deve rispettare le regole indicate dalla tabella G.2.

G.4.1 Predefinizioni di LINK

Si possono selezionare le opzioni predefite per qualsiasi informazione necessaria a LINK, in uno dei seguenti modi:

- Per selezionare la predefinizione relativa a qualunque voce della riga di comando, omettere il nome o nomi dei file precedenti la voce e digitare soltanto la virgola richiesta. L'unica eccezione è rappresentata dalla predefinizione dell'immissione *filemap*: se si utilizza una virgola come parametro formale per questa voce, LINK crea un file map.
- Per selezionare la predefinizione per qualunque prompt, premere INVIO.
- Per selezionare le predefinizioni per tutte le rimanenti voci della riga di comando o prompt, digitare un punto e virgola dopo una voce. L'unico input necessario è uno o più nomi di file oggetto.

L'elenco seguente mostra le predefinizioni che LINK utilizza per i file eseguibili, i file map, e le librerie.

Tipo di file	*Predefinizione*
Eseguibile	Nome di base del primo file oggetto fornito, seguito dall'estensione .EXE. Per assegnare un nuovo nome al file eseguibile, è necessario specificare soltanto il nuovo nome di base; se si fornisce un nome di file senza estensione, LINK aggiunge automaticamente .EXE.
Map	Il nome speciale NUL.MAP, che indica a LINK di *non* creare un file map. Per crearne uno è necessario specificare soltanto il nome di base; se si assegna al file un nome senza estensione, LINK aggiunge automaticamente l'estensione .MAP.
Librerie	Librerie specificate nei file oggetto forniti. Se si sceglie l'opzione **Genera un file EXE autonomo**, BCOM45.LIB è la libreria predefinita; altrimenti, la predefinita è BRUN45.LIB. Per specificare una libreria che non sia quella predefinita, è sufficiente il nome di base; se ad una libreria si assegna un nome senza estensione, LINK aggiunge automaticamente l'estensione .LIB. La sezione G.4.3 spiega come specificare librerie diverse da quelle predefinite.

Esempi

L'esempio seguente mostra un file risposte che indica a LINK di linkare i moduli oggetto `FRAME`, `TEXT`, `TABLE`, e `LINEOUT`. Verrà creato il file eseguibile `FRAME.EXE` ed il file map `FRAMESYM.MAP`. L'opzione `/PAUSE` fa interrompere LINK prima di generare il file eseguibile, per consentire un eventuale scambio di dischetti. L'opzione `/MAP` indica a LINK di includere nel file map simboli pubblici e indirizzi. LINK sottopone al linker anche qualsiasi routine necessaria dal file libreria `GRAF.LIB`. Le sezioni G.4.6.2 e G.4.6.11 contengono ulteriori informazioni relative alle opzioni /PAUSE e /MAP.

```
FRAME TEXT TABLE LINEOUT
/PAUSE /MAP
FRAMESYM
GRAF.LIB
```

Nell'esempio successivo, LINK carica e sottopone al linker i file oggetto `FRAME.OBJ`, `TEXT.OBJ`, `TABLE.OBJ`, e `LINEOUT.OBJ`, cercando eventuali riferimenti non trovati nel file libreria `COBLIB.LIB`. Per definizione, il file eseguibile viene chiamato `FRAME.EXE`. Viene anche generato un file map chiamato `FRAMESYM.MAP`.

```
LINK FRAME+TEXT+TABLE+LINEOUT, ,FRAMESYM, COBLIB.LIB
```

L'esempio che segue mostra come far continuare un prompt digitando un segno più (+) dopo la risposta. L'esempio esegue il linkaggio di tutti i file oggetto specificati, quindi crea un file eseguibile. Poiché, come risposta al prompt "File eseguibile", viene digitato un punto e virgola, al file eseguibile viene assegnato il nome predefinito, che corrisponde a quello di base del primo file oggetto specificato (`FRAME`), più l'estensione `.EXE`. Le predefinizioni vengono utilizzate anche per gli altri prompt. Ne consegue che non viene creato alcun file map, e che, nell'eseguire il link, vengono utilizzate come predefinite le librerie nominate nei file oggetto.

```
LINK
Moduli oggetto [.OBJ]: FRAME TEXT TABLE LINEOUT+
Moduli oggetto [.OBJ]: BASELINE REVERSE COLNUM+
Moduli oggetto [.OBJ]: ROWNUM
File eseguibile [FRAME.EXE]: ;
```

G.4.2 Specificazione dei file per LINK

Le regole per specificare i nomi dei file al linker corrispondono a quelle per specificare i nomi dei file al comando BC: le lettere maiuscole e minuscole possono essere utilizzate indistintamente ed i nomi dei file possono includere i nomi dei percorsi di ricerca per indicare a LINK di cercare o creare file nel percorso di ricerca specificato. Per ulteriori informazioni, consultare la sezione G.3.1.

G.4.3 Specificazione delle librerie per LINK

Generalmente, non è necessario comunicare a LINK il nome di una libreria autonoma. Quando il comando BC crea i file oggetto, inserisce in ciascuno di essi il nome della appropriata libreria autonoma. Quando il file oggetto viene passato al linker, LINK cerca la libreria con il nome trovato nel file oggetto e la collega automaticamente al file oggetto.

Per eseguire il link dei file oggetto con una libreria autonoma, che non sia predefinita, è necessario comunicare a LINK il nome della libreria non predefinita digitandola in uno dei seguenti modi:

- Dopo la terza virgola sulla riga del comando LINK. Una virgola segue l'elenco dei file oggetto, una l'elenco dei file eseguibili e una il file di lista. L'ultimo nome è quello della libreria.
- In risposta al prompt "Librerie" del comando LINK.

LINK cerca le librerie specificate per chiarire i riferimenti esterni prima di cercare le librerie predefinite.

Può essere opportuno eseguire il link con una libreria autonoma che non sia quella predefinita per:

- Eseguire il link con librerie autonome supplementari.
- Eseguire il link con librerie in percorsi di ricerca diversi. Se si specifica il nome completo di un percorso di ricerca di una libreria, LINK la cerca solo in quel percorso, altrimenti la cerca nei tre punti seguenti:
  - Nell'attuale directory attiva
  - In eventuali percorsi di ricerca o unità disco specificati dopo la terza virgola sulla riga del comando LINK
  - Nei punti definiti dalla variabile di ambiente LIB
- Ignorare la libreria specificata nel file oggetto. In questo caso, è necessario fornire anche l'opzione /NOD del comando LINK oltre a specificare la libreria che si vuole linkare. Per ulteriori informazioni relative a questa opzione, consultare la sezione G.4.6.8.

G.4.4 Memoria necessaria a LINK

Il comando LINK utilizza la memoria disponibile per la sessione di linkaggio. Se i file da sottoporre al linker creano un file di output che supera la memoria disponibile, LINK crea un file temporaneo che funge da memoria. Questo file viene gestito in uno dei seguenti modi, a seconda della versione del DOS:

- Per creare il file temporaneo, LINK utilizza la directory specificata dalla variabile di ambiente TMP di DOS. Ad esempio, se questa variabile fosse impostata su C:\DIRTEMP, LINK creerebbe il file temporaneo in questa directory.

  In mancanza di una variabile di ambiente TMP, o della directory specificata da TMP, LINK inserisce il file temporaneo nella directory corrente.
- Se si esegue LINK in una versione 3.0 o successiva del DOS, LINK utilizza una chiamata di sistema per creare un file temporaneo che abbia un nome nuovo nella directory dei file temporanei.
- Se si esegue LINK in una versione del DOS precedente alla 3.0, LINK crea un file temporaneo chiamato VM.TMP.

Quando il linker crea un file temporaneo appare il messaggio:

```
E' stato creato il file temporaneo filetemp
Non sostituire il disco nell'unità, lettera
```

dove *filetemp* è un ".\" seguito dal nome del file VM.TMP o da un nome creato dal DOS, mentre *lettera* è l'unità disco contenente il file temporaneo.

Il messaggio

```
Non sostituire il disco nell'unità
```

non appare, a meno che l'unità disco *lettera* non sia una unità floppy. Se viene visualizzato questo messaggio, non bisogna rimuovere il disco dall'unità disco finché non è terminata la sessione di linkaggio. Rimuovendo il disco si potrebbero verificare operazioni imprevedibili, facendo apparire il seguente messaggio:

```
fine file imprevista nel file di lavoro
```

Se si ottiene questo messaggio, è necessario riavviare la sessione di linkaggio.

Il file temporaneo creato da LINK è solo un file di lavoro, e viene cancellato alla fine della sessione.

Nota Non assegnare ad alcun file il nome VM.TMP. LINK visualizza un messaggio di errore se incontra un file con questo nome.

G.4.5 Esecuzione del link con programmi a linguaggio misto

Con il comando LINK è possibile eseguire il link con programmi a linguaggio misto. Il linkaggio di file .OBJ dall'interno di un altro linguaggio può determinare dei problemi. Presupposti diversi in altri linker possono alterare i file di QuickBASIC.

Nelle sezioni seguenti verrà discussa l'esecuzione del link con moduli scritti in linguaggio Pascal, FORTRAN, e assembler.

G.4.5.1 Moduli in Pascal e FORTRAN nei programmi QuickBASIC

I moduli compilati in Microsoft Pascal o FORTRAN possono essere linkati a programmi BASIC, come viene spiegato nella *Guida alla programmazione in linguaggi misti Microsoft*. Questi possono anche essere incorporati in librerie Quick. I programmi QuickBASIC contenenti codici compilati in Microsoft Pascal devono assegnare al Pascal uno spazio di memoria indirizzabile di almeno 2K. L'esempio seguente utilizza l'istruzione **DIM** per allocare una matrice statica di almeno 2K in un blocco comune con nome chiamato NMALLOC:

```
DIM nome%(2048) : COMMON SHARED /NMALLOC / nome%()
```

Il modulo di esecuzione in Pascal presume che sia sempre disponibile uno spazio di memoria indirizzabile di almeno 2K. Se il codice Pascal non riesce ad allocare lo spazio necessario, QuickBASIC potrebbe crollare. Ciò vale sia per il codice Pascal delle librerie Quick che per quello linkato nei file eseguibili. La situazione è simile per le operazioni di I/O del FORTRAN, che pure richiedono un buffer di memoria indirizzabile, che può essere fornito mediante un blocco comune NMALLOC.

G.4.5.2 Allocazione di matrici STATIC nelle routine in linguaggio assembler

Per passare dati di matrici statiche a routine in linguaggio assembler, è necessario utilizzare le parole chiave **SEG** o **CALLS**, o puntatori di tipo FAR. Non è possibile presupporre in quale segmento si trovino i dati. Come alternativa, si possono dichiarare tutte le matrici dinamiche (utilizzando sempre i puntatori FAR), dato che BC e l'ambiente QuickBASIC le gestiscono allo stesso modo.

G.4.5.3 Riferimenti a DGROUP nei moduli estesi in corso di esecuzione

Nei programmi in linguaggio misto che utilizzano il comando **CHAIN**, è opportuno assicurarsi che nessun codice in un modulo di esecuzione esteso contenga un riferimento a DGROUP. (Il comando **CHAIN** fa spostare DGROUP, ma non ne aggiorna i relativi riferimenti.) Questa regola vale solo per i programmi in linguaggio misto; essa può essere ignorata invece nei programmi scritti interamente in BASIC, perché le routine del BASIC non fanno mai riferimento a DGROUP.

Per evitare questo inconveniente, si può utilizzare il valore di SS, dal momento che il BASIC considera sempre uguali SS e DGROUP.

G.4.6 Utilizzo delle opzioni di LINK

Le opzioni di LINK vanno precedute dalla sbarra (/), il carattere indicante le opzioni del linker. Non si distinguono le maiuscole dalle minuscole; ad esempio, /NOI e /noi rappresentano la stessa opzione.

Le opzioni di LINK possono essere abbreviate per risparmiare spazio e lavoro. L'abbreviazione valida più breve in un'opzione è indicata nella sua sintassi. Ad esempio, diverse opzioni iniziano con le lettere "NO"; di conseguenza, perché risultino distinte, le abbreviazioni delle opzioni devono essere più lunghe di "NO". Non è possibile utilizzare "NO" come abbreviazione dell'opzione /NOIGNORECASE, perché LINK non può distinguere a quale delle opzioni che iniziano con "NO" ci si riferisce. L'abbreviazione valida più breve per questa opzione è /NOI.

Un'abbreviazione deve iniziare con la prima lettera dell'opzione ed essere continua, senza spazi o trasposizioni, fino all'ultima lettera digitata.

Alcune opzioni di LINK assumono un valore numerico; usare una delle seguenti forme:

- Un numero decimale compreso tra 0 e 65535.
- Un numero ottale compreso tra 0 e 0177777. Viene interpretato come ottale un numero che inizia con uno zero (0). Ad esempio, 10 è un numero decimale, mentre 010 è il numero ottale equivalente all'otto decimale.
- Un numero esadecimale compreso tra 0 e 0xFFFF. Viene interpretato come esadecimale un numero che inizia con uno zero seguito da una x o una X. Ad esempio, 0x10 è il numero esadecimale equivalente a 16.

Le opzioni di LINK sono valide per tutti i file elaborati dal linker, indipendentemente dal punto in cui vengono specificate.

Se si è soliti usare uno stesso insieme di opzioni per LINK, si può utilizzare la variabile di ambiente LINK di DOS per specificare una volta per tutte le opzioni da usare ogni volta che si esegue il link. Impostando questa variabile, LINK vi leggerà le opzioni, che saranno state elencate nello stesso ordine che nella riga di comando. Non è possibile specificare gli argomenti dei nomi dei file nella variabile di ambiente LINK.

Nota Un'opzione nella riga di comando annulla l'effetto di qualsiasi opzione nella variabile di ambiente con la quale è in contrasto. Ad esempio, l'opzione della riga del comando `/SE:256` annulla l'effetto dell'opzione `/SE:512`.

Per evitare che venga utilizzata un'opzione nella variabile di ambiente, è necessario reimpostare la variabile di ambiente.

Esempio

Nell'esempio che segue, viene sottoposto al linker il file `PROVA.OBJ` con le opzioni `/SE:256` e `/CO`, e poi il file `PROG.OBJ`, con l'opzione `/NOD` oltre che alla `/SE:256` e `/CO`.

```
SET LINK = /SE:256 /CO
LINK PROVA;
LINK /NOD PROG;
```

G.4.6.1 Visualizzazione dell'elenco delle opzioni (/HE)

/HE[LP]

L'opzione /HE comunica a LINK di visualizzare sullo schermo l'elenco delle opzioni disponibili.

G.4.6.2 Pausa durante l'esecuzione del link (/PAU)

/PAU[SE]

L'opzione /PAU indica a LINK di eseguire una pausa durante la sessione di linkaggio e di visualizzare un messaggio prima di creare il file eseguibile. Ciò consente di inserire un nuovo disco su cui memorizzare questo file.

Se si specifica l'opzione /PAUSE, LINK visualizza il seguente messaggio prima di creare il file eseguibile:

```
Sta per essere creato il file .EXE
Sostituire il disco nell'unità lettera e premere <INVIO>
```

lettera corrisponde all'unità disco corrente. LINK riprende l'elaborazione nel momento in cui viene premuto il tasto INVIO.

Nota Non rimuovere il disco sul quale viene creato il file di codice né quello utilizzato per il file temporaneo. Se un file temporaneo viene creato in un disco che si intende sostituire, premere la combinazione di tasti CTRL+C per terminare la sessione di linkaggio. Riordinare i file in modo che quello temporaneo e quello eseguibile possano essere memorizzati sullo stesso disco. Quindi riavviare il linker.

G.4.6.3 Visualizzazione di informazioni sull'esecuzione del link (/I)

/I[NFORMATION]

L'opzione /I visualizza informazioni relative all'esecuzione del link, incluse la fase di esecuzione e i nomi dei file oggetto che si stanno collegando.

Questa opzione consente di determinare le posizioni dei file oggetto e l'ordine di linkaggio.

G.4.6.4 Disattivazione dei solleciti durante l'esecuzione del link (/B)

/B[ATCH]

L'opzione /B indica a LINK di non richiedere l'immissione di un nuovo percorso di ricerca ogni volta che non riesce a trovare un file libreria o un file oggetto necessari. Durante l'utilizzo di quest'opzione, il linker continua semplicemente ad eseguire, senza usare il file in questione.

Quest'opzione può causare riferimenti esterni insoluti; il suo scopo fondamentale è quello di permettere l'utilizzo di file batch o MAKE per linkare molti file eseguibili con un solo comando, se non si vuole che LINK interrompa l'elaborazione in mancanza di un file richiesto. Essa è utile anche quando si reindirizza l'output di LINK per creare un file contenente il risultato del linkaggio per riferimento futuro. Tuttavia, questa opzione non evita il sollecito di eventuali argomenti mancanti dalla riga del comando di LINK.

G.4.6.5 Creazione delle librerie Quick (/Q)

/Q[UICKLIB]

L'opzione /Q indica a LINK di unire in una libreria Quick i file oggetto specificati. All'avvio dell'ambiente QuickBASIC, è possibile poi caricare la libreria Quick con l'opzione /L nella riga del comando QB. Se si utilizza l'opzione /Q, è necessario assicurarsi di specificare il file BQLB45.LIB nell'elenco delle librerie, per includere le routine QuickBASIC per il supporto delle librerie Quick.

Per ulteriori informazioni relative alla creazione e al caricamento delle librerie Quick, consultare l'appendice H, "Creazione ed utilizzo delle librerie Quick".

Nota Non è possibile utilizzare l'opzione /EXEPACK con l'opzione /Q.

G.4.6.6 Compattazione dei file eseguibili (/E)

/E[XEPACK]

L'opzione /E rimuove le sequenze di byte ripetuti (solitamene caratteri nulli) ed ottimizza la "tabella di reindirizzo memoria durante il caricamento" prima di creare il file eseguibile. Si tratta, in sostanza, di una tabella di riferimenti tutti relativi all'inizio del programma. Ciascun riferimento poi cambia quando l'immagine eseguibile viene caricata in memoria, e viene assegnato un indirizzo reale all'istruzione iniziale.

Nota I file eseguibili sottoposti al linker con questa opzione possono risultare più piccoli ed essere caricati più velocemente rispetto a quelli linkati senza questa opzione.

G.4.6.7 Disattivazione della compattazione dei segmenti (/NOP)

/NOP[ACKCODE]

L'opzione /NOP generalmente non è necessaria poiché la compattazione dei segmenti di codice è normalmente disattivata. Tuttavia, se una variabile di ambiente DOS, come LINK, attiva automaticamente la compattazione, è possibile utilizzare questa opzione per disattivarla nuovamente.

G.4.6.8 Disattivazione dell'uso delle normali librerie BASIC (/NOD)

/NOD[EFAULTLIBRARYSEARCH]

Quando il comando BC crea un file oggetto, vi include i nomi delle librerie "standard" – le librerie richieste da LINK per risolvere i riferimenti esterni. L'opzione /NOD indica a LINK di *non* cercare nelle librerie nominate nei file oggetto per chiarire tali riferimenti esterni.

Generalmente, i programmi QuickBASIC non funzionano correttamente senza le librerie standard (BRUN45.LIB e BCOM45.LIB). Perciò, se si utilizza l'opzione /NOD, è opportuno fornire esplicitamente il nome del percorso di ricerca della libreria standard richiesta.

G.4.6.9 Disattivazione dell'uso dei dizionari (/NOE)

/NOE[XTDICTIONARY]

Se LINK sospetta che sia stato ridefinito un simbolo pubblico, esso sollecita la riesecuzione del link con l'opzione /NOE. In seguito, per risolvere le contraddizioni, cerca nei singoli file oggetto, piuttosto che nei "dizionari" da esso stesso creati. L'opzione /NOE si deve utilizzare, ad esempio, per linkare un programma con 87.LIB.

G.4.6.10 Impostazione di un numero massimo di segmenti (/SE)

/SE[GMENTS]:*numero*

L'opzione /SE controlla il numero massimo di segmenti ammessi da LINK in un programma. Il valore predefinito è 128, ma è possibile impostare *numero* ad un qualunque valore (specificandolo in decimale, ottale, o esadecimale) nell'intervallo 1–1024 (decimale).

Per ciascun segmento, LINK deve riservare dello spazio memoria per restare al corrente delle informazioni del segmento. Impostando il limite del numero dei segmenti ad un valore superiore a 128, LINK riserverà più spazio per le informazioni dei segmenti. Nei programmi contenenti meno di 128 segmenti, è possibile ridurre al minimo la quantità di memoria necessaria a LINK, impostando *numero* al valore corrispondente al numero dei segmenti del programma. LINK visualizza un messaggio di errore se questo numero è troppo elevato rispetto alla quantità di memoria disponibile.

G.4.6.11 Creazione di un file map (/M)

/M[AP]

L'opzione /M crea un file map, che elenca i segmenti e i simboli pubblici di un programma. LINK cerca sempre di riservare tutta la memoria disponibile per le operazioni di ordinamento di questi simboli. Se il loro numero supera il limite dello spazio memoria, LINK genera un elenco non ordinato. Il file map *filemap* contiene un elenco di simboli ordinati per indirizzo; non contiene però un elenco ordinato per nome. Segue un file map di esempio:

```
Start   Stop    Length Name                  Class
 00000H 01E9FH 01EA0H _TEXT                  CODE
 01EA0H 01EA0H 00000H C_ETEXT                ENCODE
.
.
.
```

Le informazioni contenute nelle colonne `Start` e `Stop` mostrano l'indirizzo a 20 bit (in esadecimale) di ciascun segmento, relativo all'inizio del modulo caricato, che comincia in posizione zero. La colonna `Length` riporta la lunghezza del segmento in byte, la colonna `Name` indica il nome del segmento, e la colonna `Class` le informazioni relative al tipo del segmento. Per ulteriori informazioni relative a gruppi, segmenti e classi, consultare il manuale *Microsoft MS-DOS Programmer's Reference.*

L'indirizzo iniziale ed il nome di ciascun gruppo seguono l'elenco dei segmenti. Di seguito viene presentato un listato di esempio di un gruppo:

```
Origin   Group
01EA:0   DGROUP
```

In questo esempio, DGROUP è il nome del gruppo di dati.

Il file map illustrato di seguito contiene due elenchi di simboli globali: il primo è ordinato per nome del simbolo secondo la tabella dei caratteri ASCII; l'altro per indirizzo. L'annotazione `Abs` appare accanto ai nomi dei simboli assoluti (simboli contenenti valori di costanti a 16 bit non relativi agli indirizzi del programma).

Molti simboli globali presenti nel file map vengono utilizzati internamente dal compilatore e dal linker. Essi di solito iniziano con i caratteri `B$` o terminano con `QQ`.

```
Address          Publics by Name

  01EA:0096        STKHQQ
  0000:1D86        B$Shell
  01EA:04B0        _edata
  01EA:0910        _end
.
.
.
  01EA:00EC        _abrkp
  01EA:009C        _abrktb
  01EA:00EC        _abrktbe
  0000:9876   Abs  _acrtmsg
  0000:9876   Abs  _acrtuseb
.
.
.
  01EA:0240        _argc
  01EA:0242        _argv

  Address         Publics by Value

  0000:0010        _main
  0000:0047        _htoi
.
.
.
```

Gli indirizzi dei simboli esterni sono forniti nel formato *blocco:offset*, ed indicano la posizione del simbolo relativo allo zero (l'inizio del modulo caricato).

Dopo gli elenchi dei simboli, il file map riporta il punto iniziale del programma, come illustra l'esempio che segue:

```
Program entry point at 0000:0129
```

Un file map può inoltre essere specificato fornendo un nome di file map alla riga di comando LINK o in risposta al prompt "File di lista".

G.4.6.12 Inserimento dei numeri di riga in un file map (/LI)

/LI[NENUMBERS]

L'opzione /LI crea un file map e vi include i numeri di riga e i relativi indirizzi del programma sorgente. Se si sta compilando ed eseguendo il link in due fasi distinte, questa opzione è valida soltanto se si esegue il link di file oggetto compilati con l'opzione /M.

G.4.6.13 Compattazione dei segmenti contigui (/PAC)

/[NO]PAC[KCODE][:*numero*]

L'opzione /PAC indica a LINK di raggruppare i segmenti di codice contigui. I segmenti di codice che si trovano nello stesso gruppo condividono lo stesso indirizzo di segmento; gli indirizzi di offset vengono quindi aumentati a seconda del caso. Ne consegue che molte istruzioni, che altrimenti avrebbero indirizzi di segmento diversi, condividono lo stesso indirizzo.

Quando specificato, *numero* indica il limite della dimensione dei gruppi generati dall'opzione /PAC e impedisce l'aggiunta di segmenti ad un gruppo se ciò fa oltrepassare *numero*. A quel punto, LINK inizia a creare un nuovo gruppo con i segmenti di codice rimanenti. Se non viene indicato il valore di *numero*, quello predefinito è 65530.

Sebbene LINK non esegua la compattazione di segmenti contigui a meno che non venga richiesto esplicitamente, è possibile utilizzare l'opzione /NOPACKCODE per disattivare la compattazione dei segmenti se, ad esempio, nella variabile di ambiente LINK di DOS è stata specificata l'opzione /PAC.

G.4.6.14 Utilizzo del debugger CodeView (/CO)

/CO[DEVIEW]

L'opzione /CO prepara un file eseguibile per la messa a punto mediante il debugger CodeView. Se si sta compilando ed eseguendo il link in due fasi distinte, quest'opzione è valida soltanto se si sta collegando file oggetto compilati con l'opzione /ZI del comando BC. Analogamente, non bisogna utilizzarla insieme all'opzione /Q del comando LINK, perché una libreria Quick non può essere messa a punto con il debugger CodeView.

G.4.6.15 Distinzione tra maiuscole e minuscole (/NOI)

/NOI[GNORECASE]

L'opzione /NOI indica a LINK di distinguere le lettere maiuscole dalle minuscole; ad esempio, LINK potrebbe considerare i nomi ABC, abc, e Abc come tre nomi diversi. Durante l'esecuzione del link, non specificare l'opzione /NOI sulla riga del comando LINK.

G.4.7 Altre opzioni della riga di comando di LINK

Non tutte le opzioni del comando LINK sono utilizzabili con i programmi QuickBASIC. Quelle descritte di seguito possono essere usate con programmi Microsoft QuickBASIC, ma non è mai necessario specificarle in quanto sono eseguite automaticamente dal comando BC o da QuickBASIC:

Opzione	*Azione*
/CP[ARMAXALLOC]:*numero*	Imposta il numero massimo di paragrafi a 16 byte richiesti dal programma quando questo viene caricato in memoria a *numero*, dove *numero* è un intero compreso tra 1 e 65535. Il sistema operativo utilizza questo valore quando assegna spazio al programma, prima di caricarlo. Sebbene sia possibile utilizzare questa opzione nella riga del comando LINK, essa non ha effetto, perché il programma BASIC durante la sua esecuzione controllerà la quantità di memoria disponibile.
/DO[SSEG]	Garantisce che i segmenti vengano ordinati secondo le predefinizioni per i linguaggi Microsoft evoluti. I programmi QuickBASIC utilizzano sempre questo ordine per predefinizione.
/ST[ACK]:*numero*	Specifica la dimensione dello stack del programma, dove *numero* è un qualunque valore positivo (decimale, ottale o esadecimale) fino a 65535 (decimale) e rappresenta la dimensione dello stack in byte. La libreria standard BASIC imposta a 2K la dimensione predefinita dello stack.
/DS[ALLOCATE]	Carica tutti i dati a partire dall'estremità superiore del segmento di dati predefinito.
/HI[GH]	Posiziona il file eseguibile nella più alta posizione di memoria disponibile.
/NOG[ROUPASSOCIATION]	Indica a LINK di ignorare le associazioni di gruppi quando assegna indirizzi agli elementi dei dati e del codice.
/O[VERLAYINTERRUPT]:*numero*	Specifica un numero di interrupt diverso da 0x3F per passare il controllo agli overlay.

Nota Non utilizzare le opzioni /DS, /HI, /NOG, e /O nel linkaggio di file oggetto compilati da BC. Esse sono utilizzabili solo con file oggetto creati dal Microsoft Macro Assembler (MASM).

G.5 Gestione delle librerie autonome: LIB

Il Microsoft Library Manager (LIB) gestisce il contenuto delle librerie autonome. Una libreria autonoma è costituita da "moduli oggetto" – ovvero file oggetto uniti tra loro per creare una libreria. A differenza di un file oggetto, un modulo oggetto non può esistere indipendentemente dalla sua libreria, e non dispone di un nome di percorso di ricerca o di una estensione relativa al nome del proprio file. Utilizzando LIB, è possibile:

- Unire dei file oggetto per creare una nuova libreria
- Aggiungere dei file oggetto ad una libreria già esistente
- Eliminare o sostituire dei moduli oggetto di una libreria già esistente
- Estrarre dei moduli oggetto da una libreria esistente e collocarli in file oggetto separati
- Unire il contenuto di due librerie già esistenti in una nuova

Quando si aggiorna una libreria esistente, LIB esegue tutte le proprie operazioni su una copia della libreria. Questo meccanismo permette di avere una copia backup di qualsiasi libreria aggiornata in caso di problemi con la versione aggiornata.

G.5.1 Esecuzione di LIB

E' possibile fornire input al comando LIB in uno dei seguenti modi:

- Specificando l'input sulla riga di comando nella forma seguente:

  LIB *vecchialib* [/P[AGESIZE]:*numero*] [*comandi*] [, [*filelista*] [, [*nuovalib*]]] [;]

  La riga di comando può contenere massimo 128 caratteri.

continua

- Digitando

  ```
  lib
  ```

  e rispondendo ai seguenti prompt:

  ```
  Nome della libreria:
  Operazioni:
  File di lista:
  Libreria di output:
  ```

  Per fornire più file ad un prompt, digitare una & (e commerciale) alla fine della riga. Il prompt riapparirà sulla riga successiva.

- Impostando un file risposte contenente le risposte ai prompt del comando LIB, quindi digitando il comando LIB nella forma seguente:

  `LIB @`*nomefile*

  in cui *nomefile* è il nome del file risposte. E' necessario che le risposte si trovino nello stesso ordine dei prompt di LIB sopra trattati. E' anche possibile digitare il nome di un file risposte dopo qualunque prompt di LINK, o in un punto qualunque della riga di comando di LIB.

La tabella G.3 mostra l'input da fornire sulla riga del comando LIB, o in risposta a ciascun prompt. Se si sta utilizzando un file risposte, è necessario che ciascuna risposta segua le regole delineate in questa tabella.

Tabella G.3 Input al comando LIB

Campo	***Prompt***	***Input***
vecchialib	"Nome della libreria"	Nome della libreria che si sta modificando o creando. Se essa non esiste, LIB chiede se la si vuole creare: digitare S per crearla o N per terminare LIB. Questo messaggio non appare se si digitano caratteri di comando, una virgola o un punto e virgola dopo il nome della libreria. Un punto e virgola indica a LIB di eseguire una verifica della libreria; in questo caso, viene visualizzato un messaggio se sono stati riscontrati errori in un modulo della libreria.

Campo	***Prompt***	***Input***
/P:*numero*	/P:*numero* dopo il prompt "Nome della libreria"	La dimensione di pagina della libreria, che viene impostata a *numero*; *numero* è una potenza intera di 2 compresa tra 16 e 32768, inclusi. La dimensione di pagina predefinita per una nuova libreria è 16 byte. La disposizione dei moduli della libreria è sempre tale che ogni modulo inizia un multiplo della dimensione di pagina (in byte) dopo l'inizio del file.
/I	nessuno	Indica a LIB di non distinguere tra maiuscole e minuscole nel confronto dei simboli (predefinita). E' necessario utilizzarla per l'unione con librerie che rilevano le maiuscole.
/NOE	nessuno	Indica a LIB di non generare un dizionario esteso.
/NOI	nessuno	Indica a LIB di distinguere tra maiuscole e minuscole nel confronto dei simboli (l'opzione poi rimane attiva).
comandi	"Operazioni"	Simboli di comando e file oggetto che indicano a LIB quali modifiche apportare alla libreria.
filelista	"File di lista"	Nome di un file di lista dei rimandi interni. Non viene creato alcun file elenco se non si indica un nome di file.
nuovalib	"Libreria di output"	Nome della libreria modificata, creata da LIB come output. Se non si indica un nuovo nome di libreria, la libreria originaria, rimasta inalterata, viene memorizzata in un file libreria con lo stesso nome, ma con estensione .BAK invece di .LIB.

G.5.2 Impostazioni predefinite di LIB

LIB dispone di impostazioni proprie (predefinite). E' possibile scegliere tali impostazioni per le informazioni necessarie a LIB in uno dei seguenti modi:

- Per selezionare l'impostazione predefinita di qualunque voce sulla riga di comando, omettere il nome o i nomi dei file prima della voce, e digitare solo la virgola richiesta. L'unica eccezione è costituita dall'impostazione predefinita per la voce *filelista*: se si omette questa voce, LIB crea un file di lista dei rimandi interni.
- Per selezionare la predefinizione per un prompt, premere INVIO.
- Per selezionare le predefinizioni per tutte le voci della riga di comando e per i prompt successivi ad una voce o un prompt, digitare, dopo la voce o il prompt, un punto e virgola (;) come ultimo carattere della riga di comando.

L'elenco seguente presenta le predefinizioni che LIB utilizza per i file di lista dei rimandi interni e per le librerie di output:

File	***Predefinizione***
File di lista dei rimandi interni	Il nome speciale di file NUL, che indica al linker di *non* creare un file di lista dei rimandi interni.
Libreria di output	La voce *vecchialib* o la risposta al prompt "Nome della libreria".

G.5.3 File di lista dei rimandi interni

Un file di lista dei rimandi interni indica quali routine sono contenute in una libreria autonoma e i file oggetto originari dai quali sono state ottenute, e contiene gli elenchi seguenti:

- Un elenco in ordine alfabetico di tutti i simboli pubblici della libreria. Il nome di ciascun simbolo è seguito da quello del modulo in cui è stato definito.
- Un elenco dei moduli della libreria. Sotto il nome di ciascun modulo si trova un listato in ordine alfabetico dei simboli pubblici definiti in quel modulo.

G.5.4 Simboli di comando

Per indicare a LIB le modifiche che si vogliono apportare ad una libreria, digitare un simbolo di comando, quale +, −, −+, * o −*, seguito immediatamente dal nome di un modulo, di un file, o di una libreria. E' possibile specificare più operazioni, in qualsiasi ordine.

L'elenco seguente presenta ciascun simbolo di comando LIB, il tipo di nome del file da specificare con quel simbolo, e la funzione del simbolo:

Comando	***Significato***
+{*fileoggetto* | *lib*}	Aggiunge alla libreria di input il file oggetto specificato, rendendolo l'ultimo modulo della libreria, se il segno + è seguito dal nome di un file oggetto. E' possibile utilizzare come nome un percorso di ricerca. Poiché LIB assegna automaticamente l'estensione .OBJ, non è necessario digitarla. Se il segno + è invece seguito dal nome di una libreria con estensione .LIB, il suo contenuto viene aggiunto alla libreria di input. L'estensione .LIB in questo caso è obbligatoria.
−*modulo*	Elimina il modulo specificato dalla libreria di input. Il nome del modulo non può includere un percorso di ricerca o estensione.
−+*modulo*	Sostituisce il modulo specificato nella libreria di input. I nomi dei moduli non devono includere percorso di ricerca né estensione. LIB elimina il modulo specificato e poi aggiunge il file oggetto con lo stesso nome del modulo. Il file oggetto deve avere l'estensione .OBJ e trovarsi nella directory corrente.
**modulo*	Copia il modulo specificato dalla libreria in un file oggetto nella directory corrente. Il modulo resta nel file libreria. Quando LIB copia il modulo nel file oggetto, vi aggiunge l'estensione .OBJ. Non è possibile ridefinire l'estensione (.OBJ), l'unità disco, o il percorso che LIB assegnerà al nuovo file oggetto; è possibile, però, assegnare al file un nuovo nome o copiarlo in un altro punto dopo la sua creazione.
−**modulo*	Sposta il modulo oggetto specificato dalla libreria in un file oggetto. Questa operazione equivale alla copia del modulo in un file oggetto, come descritto qui sopra, seguita dalla sua eliminazione dalla libreria.

Esempi

L'esempio seguente utilizza il simbolo di comando della sostituzione (−+), per indicare a LIB di sostituire il modulo `HEAP` nella libreria `LANG.LIB`. LIB elimina `HEAP` ed aggiunge come nuovo modulo il file oggetto `HEAP.OBJ`. Il punto e virgola alla fine della riga di comando indica a LIB di utilizzare le impostazioni predefinite per i rimanenti prompt. Ciò significa che non viene creato alcun file di lista e che, invece di creare un nuovo file libreria, le modifiche vengono apportate al file libreria originario.

```
LIB LANG-+HEAP;
```

Gli esempi seguenti eseguono la stessa funzione del primo, ma in due operazioni distinte, utilizzando i simboli di comando di addizione (+) e di sottrazione (–). L'effetto è lo stesso perché le operazioni di eliminazione vengono sempre portate a termine prima di quelle di aggiunta, indipendentemente dal loro ordine sulla riga di comando. Quest'ordine di esecuzione evita confusione quando si sostituisce la versione vecchia di un modulo in un file libreria con una nuova.

```
LIB LANG-HEAP+HEAP;
LIB LANG+HEAP-HEAP;
```

Gli esempi successivi fanno eseguire a LIB una verifica del file libreria `FOR.LIB`. Non viene eseguita alcun'altra azione. LIB visualizza qualsiasi errore di concordanza riscontrato e ritorna al sistema operativo.

```
LIB FOR;
```

L'esempio seguente indica a LIB di eseguire una verifica del file libreria `LANG.LIB` e di creare un file di lista dei rimandi interni detto `LRIMINT.PUB`.

```
LIB LANG,LRIMINT.PUB
```

Nell'esempio seguente, LIB sposta il modulo `VARI` dalla libreria `PRIMA.LIB` in un file oggetto chiamato `VARI.OBJ`. Durante il procedimento, il modulo `VARI` viene tolto dalla libreria. Il modulo `ALTRI` viene copiato dalla libreria al file oggetto `ALTRI.OBJ` e qui rimane. La libreria revisionata viene chiamata `SECONDA.LIB` e contiene tutti i moduli della libreria `PRIMA.LIB` tranne `VARI` che è stato rimosso mediante il simbolo di comando (`-*`). La libreria originaria, `PRIMA.LIB`, rimane inalterata.

```
LIB PRIMA -*VARI *ALTRI, ,SECONDA
```

Il contenuto del seguente file risposte fa eliminare il modulo `HEAP` dal file libreria `LIBFOR.LIB`, fa copiare (senza eliminarlo) il modulo `FOIBLES` al file oggetto `FOIBLES.OBJ`, e fa aggiungere i file oggetto `CURSOR.OBJ` e `HEAP.OBJ` come ultimi due moduli della libreria. Infine, LIB crea il file di lista dei rimandi interni `RIMINLST`.

```
LIBFOR
+CURSOR+HEAP-HEAP*FOIBLES
RIMINLST
```

G.5.5 Opzioni di LIB

LIB dispone di quattro opzioni, che si specificano sulla riga di comando dopo il nome del file libreria richiesto e prima degli eventuali comandi.

G.5.5.1 Nessuna distinzione tra maiuscole e minuscole nei simboli

/I[GNORECASE]

L'opzione /I indica a LIB di non distinguere tra maiuscole e minuscole nei simboli; è predefinita. Utilizzare questa opzione per l'unione di una libreria contrassegnata dall'opzione /NOI (sotto descritta) ad altre non contrassegnate se si desidera che la nuova libreria non sia contrassegnata.

G.5.5.2 Disattivazione dell'uso dei dizionari estesi

/NOE[XTDICTIONARY]

L'opzione /NOE indica a LIB di non creare un dizionario esteso, una parte supplementare della libreria che velocizza il linkaggio delle librerie.

Se si riscontrano gli errori `U1171` o `U1172`, o se il dizionario esteso provoca dei problemi a LINK, utilizzare l'opzione /NOE. Per ulteriori informazioni sull'utilizzo del dizionario esteso da parte di LINK, consultare la sezione G.4.6.9.

G.5.5.3 Distinzione tra maiuscole e minuscole nei simboli

/NOI[GNORECASE]

L'opzione /NOI indica a LIB di distinguere tra minuscole e maiuscole nei simboli messi a confronto, al contrario della predefinizione. Utilizzando questa opzione è possibile avere nella stessa libreria simboli diversi che differiscono solo per quanto riguarda il maiuscolo/minuscolo dei caratteri, come SPLINE e Spline.

Notare che durante la creazione di una libreria con l'opzione /NOI, LIB contrassegna internamente questa libreria per segnalare che l'opzione /NOI è attiva. La precedente versione di LIB non utilizzava questo metodo. Se si uniscono più librerie, e una di esse è contrassegnata con /NOI, anche la libreria risultante verrà contrassegnata con /NOI.

G.5.5.4 Impostazione delle dimensioni di pagina

/P[AGESIZE]:*numero*

La dimensione di pagina di una libreria influisce sull'allineamento dei moduli in essa memorizzati. Essi vengono sempre disposti dall'inizio del file, in modo da iniziare in una posizione che è un multiplo della dimensione di pagina (in byte). La dimensione di pagina predefinita di una libreria appena creata è di 16 byte.

E' possibile impostare una diversa dimensione di pagina per la libreria durante la sua creazione, oppure modificare quella di una libreria già esistente, digitando la seguente opzione dopo *vecchialib* sulla riga del comando LIB, o dopo il nome digitato in risposta al prompt "Nome di libreria":

/P[AGESIZE]:*numero*

numero specifica la nuova dimensione di pagina della libreria. E' necessario che esso sia un valore intero rappresentante una potenza di 2 compresa tra 16 e 32768.

La dimensione di pagina della libreria determina il numero dei moduli che essa può contenere. Aumentando la dimensione di pagina è possibile inserire un maggior numero di moduli nella libreria. Tuttavia, più grande è la dimensione di pagina, maggiore è la quantità di spazio memoria sprecato dalla libreria (in media, *dimpagina*/2 byte). Nella maggior parte dei casi, è opportuno utilizzare una piccola dimensione di pagina, a meno che non sia necessario inserire nella libreria numerosi moduli.

La dimensione di pagina determina anche la dimensione massima della libreria. Questo limite è dato da *numero* * 65536. Ad esempio, se si richiama LIB con l'opzione `/P:16`, la dimensione della libreria deve essere minore di un megabyte (16 * 65536 byte).

Appendice H
Creazione ed utilizzo delle librerie Quick

Quest'appendice descrive come creare e aggiornare le librerie Quick dall'interno dell'ambiente di programmazione QuickBASIC. Una libreria è un file che include il contenuto di vari moduli. A lettura ultimata, l'utente saprà:

- Costruire librerie dall'interno dell'ambiente QuickBASIC
- Caricare una libreria Quick durante l'esecuzione di un programma QuickBASIC
- Visualizzare il contenuto di una libreria Quick
- Aggiungere ad una libreria Quick routine scritte in altri linguaggi

H.1 Tipi di libreria

QuickBASIC fornisce gli strumenti per la creazione di due diversi tipi di libreria, identificati da estensioni diverse:

Estensione	***Funzione***
.QLB	L'estensione .QLB caratterizza una libreria Quick, ossia un particolare tipo di libreria che consente di aggiungere con facilità al programma le procedure di uso frequente. Una libreria Quick può contenere procedure scritte in QuickBASIC o in altri linguaggi Microsoft, come il C.
.LIB	L'estensione .LIB caratterizza una libreria autonoma (.LIB), creata dal Microsoft Library Manager, LIB. Nel costruire una libreria Quick, QuickBASIC genera contemporaneamente una libreria .LIB contenente le stesse procedure in forma leggermente diversa.

Entrambi i tipi di libreria possono essere generati dall'interno dell'ambiente di programmazione o dalla riga di comando. Si può considerare una libreria Quick come un gruppo di procedure aggiunte a QuickBASIC al momento del suo avvio. Le librerie con estensione .LIB sono sostanzialmente delle procedure compilate in modo autonomo e possono essere aggiunte ad una libreria Quick o collegate ad un modulo principale per creare un file eseguibile dalla riga di comando del DOS.

Quest'appendice contiene alcune informazioni sui programmi di utilità della riga di comando, ma l'appendice G, "Compilazione ed esecuzione del link da DOS", contiene dettagli più ampi su questo argomento.

H.2 Vantaggi delle librerie Quick

Le librerie Quick facilitano lo sviluppo e la manutenzione dei programmi. Man mano che si sviluppa un programma e le procedure assumono la loro forma definitiva, queste si possono raggruppare in una libreria Quick, tenendone da parte i codici sorgenti per eventuali modifiche o aggiornamenti futuri. Si può poi caricare la libreria all'avvio di QuickBASIC e il programma avrà accesso immediato a tutte le procedure in essa contenute.

Le procedure di una libreria Quick si comportano come istruzioni interne di QuickBASIC. Se opportunamente dichiarata, una procedura **SUB** in una libreria Quick può essere richiamata anche in assenza dell'istruzione **CALL**. Per ulteriori informazioni relative alla chiamata di una procedura **SUB** con o senza la parola chiave **CALL**, consultare il capitolo 2, "Procedure SUB e FUNCTION".

Le procedure contenute in una libreria Quick possono essere eseguite direttamente dalla finestra di esecuzione immediata, esattamente come le istruzioni BASIC. Vale a dire che se ne possono provare gli effetti prima di utilizzarle in altri programmi.

Se si sviluppano programmi insieme ad altri programmatori, le librerie Quick facilitano l'aggiornamento di un gruppo di procedure comuni. Una libreria di procedure originali fornita per uso commerciale, permetterà ai programmatori QuickBASIC di potenziare il proprio lavoro. Il programmatore può affiggere su una bacheca elettronica (bulletin board) una propria libreria, affiché questa possa essere esaminata prima dell'eventuale acquisto. Dato che le librerie Quick non contengono codice sorgente e possono essere usate solo dall'interno dell'ambiente di programmazione QuickBASIC, gli interessi del proprietario sono salvaguardati mentre aumentano le possibiltà di commercializzazione.

Nota Le librerie Quick hanno la stessa funzione delle librerie dell'utente delle versioni 2.0 e 3.0 di QuickBASIC. Una libreria dell'utente però, non può essere caricata come libreria Quick, e sarà necessario rigenerarla dal codice sorgente.

H.3 Creazione di una libreria Quick

Una libreria Quick contiene automaticamente tutti i moduli, principali e non, presenti nell'ambiente QuickBASIC al momento della propria creazione. Essa include inoltre il contenuto di qualsiasi altra libreria Quick caricata al momento dell'avviamento di QuickBASIC. Se si carica un programma intero di cui si vogliono includere nella libreria soltanto alcuni moduli, bisogna esplicitamente chiudere quelli non richiesti. I moduli possono essere chiusi con il comando **Chiudi file** del menu **File**.

Si possono determinare rapidamente i moduli caricati, esaminando la casella di riepilogo del comando **SUBroutine** del menu **Visualizza**. Questo comando, però, non visualizza le procedure contenute in una libreria caricata. Il programma di utilità (QLBDUMP.BAS), descritto nella sezione H.4.3, "Visualizzazione del contenuto di una libreria Quick", permette di elencare tutte le procedure presenti in una libreria.

In una libreria si possono includere solo moduli interi. Cioè, non si può selezionare una sola procedura tra le tante contenute in un modulo. Se si vogliono invece immettere soltanto determinate procedure del modulo, inserirle in un modulo separato, e quindi inserire quel modulo nella libreria.

Una libreria Quick deve essere autonoma. Una sua procedura può chiamare solo altre procedure all'interno della stessa libreria.

Si può ridurre il tempo di caricamento dei programmi ampi, inserendo il maggior numero possibile di routine nelle librerie Quick. L'inserimento di molte routine nelle librerie è inoltre vantaggioso se si decide di creare dal programma un file eseguibile autonomo, dal momento che i contenuti delle librerie vengono collegati senza necessità di ricompilazione.

Nota Il modulo principale può contenere o meno delle procedure. Se ne contiene e queste vengono incorporate nella libreria, viene incorporato anche l'intero modulo principale. Ciò non provoca un messaggio di errore; tuttavia il codice a livello di modulo nella libreria non sarà mai eseguito a meno che una delle sue procedure non contenga una routine (quale **ON ERROR**) che trasferisca esplicitamente il controllo al livello del modulo. Anche in questo caso, gran parte del codice a livello di modulo rimarrà inutilizzabile. Se si dispongono le procedure in moduli generalmente utilizzati insieme, le librerie Quick conterranno meno codice superfluo.

H.3.1 File necessari per la creazione di una libreria Quick

Prima di creare una libreria Quick, bisogna assicurarsi che siano disponibili i file necessari. Se non si possiede un'unità a disco rigido, si renderà necessario memorizzare i file e gli altri programmi su diversi dischi flessibili. Quando QuickBASIC non riesce a trovare un file, chiede il nome di un percorso di ricerca; in questo caso, inserire il disco appropriato e rispondere al prompt.

Assicurarsi che i seguenti file si trovino nella directory corrente o che siano rintracciabili da QuickBASIC mediante gli appositi comandi DOS:

File	*Scopo*
QB.EXE	Dirige il processo di creazione di una libreria Quick. Se si lavora solo con moduli QuickBASIC, questo si può fare in una sola fase dall'interno dell'ambiente QuickBASIC.
BC.EXE	Crea i file oggetto dal codice sorgente.
LINK.EXE	Esegue il link dei file oggetto.
LIB.EXE	Gestisce le librerie autonome di moduli oggetto.
BQLB45.LIB	Fornisce le routine richieste dalle librerie Quick. BQLB45.LIB è una libreria autonoma che viene collegata agli oggetti della nuova libreria per formare una libreria Quick.

H.3.2 Creazione di una libreria Quick

Quasi sempre le librerie Quick vengono create dall'interno dell'ambiente QuickBASIC. A volte, può rendersi necessario aggiornare una libreria o includervi routine in altri linguaggi Microsoft. In questi casi, inizialmente creare una libreria di base iniziale con le routine non BASIC dall'esterno dell'ambiente, usando direttamente LINK e LIB. Poi aggiungere le versioni più recenti dei moduli in QuickBASIC dall'interno dell'ambiente QuickBASIC.

H.3.3 Creazione di una libreria Quick dall'interno dell'ambiente

Nella creazione di una libreria Quick dall'interno dell'ambiente QuickBASIC, prima valutare se è meglio crearne una nuova, o se basta aggiornarne una esistente. Nel secondo caso, avviare QuickBASIC con l'opzione /L della riga di comando, fornendo quale argomento della riga di comando il nome della libreria da aggiornare. Allo stesso tempo, si può includere il nome del programma i cui moduli si vogliono inserire nella libreria. In questo caso, QuickBASIC carica tutti i moduli specificati nel file .MAK del programma.

H.3.3.1 Chiusura di file non richiesti

Se si carica il programma durante l'avviamento di QuickBASIC, è necessario chiudere qualsiasi modulo non richiesto nella libreria, incluso il modulo principale (a meno che esso non contenga delle procedure da inserire nella libreria).

Per chiudere i moduli, rispettare queste fasi:

1. Selezionare il comando **Chiudi file** dal menu **File**.
2. Selezionare il modulo da chiudere nella casella di riepilogo, quindi premere INVIO.
3. Ripetere le prime due operazioni finché non siano stati chiusi tutti i moduli non richiesti.

H.3.3.2 Caricamento dei file richiesti

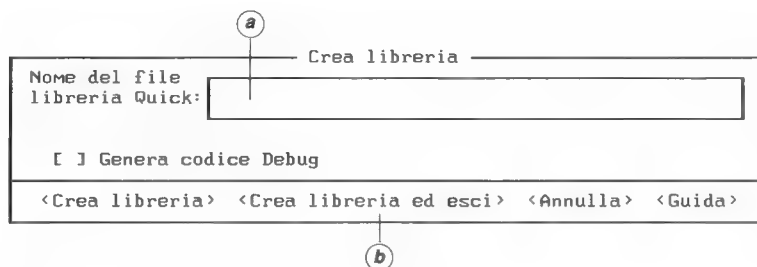
In alternativa, si può semplicemente avviare QuickBASIC, con o senza una specifica di libreria, e caricare i moduli richiesti uno per volta dall'interno dell'ambiente. In questo caso, per caricare ciascun modulo si utilizza il comando **Carica file** dal menu **File**.

Per caricare un modulo per volta in QuickBASIC:

1. Selezionare il comando **Carica file** del menu **File**.
2. Selezionare il nome del modulo da caricare nella casella di riepilogo.
3. Ripetere le operazioni dei punti 1 e 2 finché non siano stati caricati tutti i moduli richiesti.

H.3.3.3 Creazione di una libreria Quick

Una volta caricata la libreria precedente (se esiste) e tutti i moduli da includere nella libreria Quick, selezionare il comando **Crea libreria** dal menu **Esegui**. Apparirà la finestra di dialogo che si vede nell'illustrazione H.1.

Illustrazione H.1 Finestra di dialogo del comando Crea libreria

a) Digitare il nome della libreria Quick.
b) Crea la libreria ed esce al DOS.

Per creare una libreria Quick rispettare le fasi seguenti:

1. Digitare il nome della libreria da creare nella casella di testo **Nome del file libreria Quick**.

   Se viene digitato soltanto il nome di base (senza l'estensione) QuickBASIC aggiunge automaticamente l'estensione .QLB. Se si desidera che la propria libreria non abbia alcuna estensione, aggiungere al nome di base un punto (.). Altrimenti, si può digitare qualunque nome di base ed estensione si desideri (tranne che il nome di una libreria Quick già caricata), in conformità con le regole DOS sui nomi dei file.
2. Scegliere la casella di selezione del sottocomando **Genera codice Debug** solo se si sta effettivamente cercando di rilevare un presunto errore nella libreria che si sta aggiornando. Ciò ingrandisce la libreria, rallenta l'esecuzione del programma e permette di controllare quasi esclusivamente gli errori relativi ai limiti delle matrici.
3. Creare la libreria Quick:
   - Scegliere l'opzione **Crea libreria** se, dopo la creazione della libreria Quick, si desidera rimanere in ambiente QuickBASIC.
   - Scegliere l'opzione **Crea libreria ed esci**, se, dopo la creazione della libreria Quick, si vuole ritornare in ambiente DOS.

Nota Ricordarsi, quando si crea una libreria Quick che potrà essere utilizzata con un modulo non di libreria che deve gestire eventi quali le pressioni di tasto, che almeno uno dei moduli della libreria deve contenere un'istruzione di gestione degli eventi. Quest'istruzione può anche essere una semplice istruzione **TIMER OFF**, ma senza di essa gli eventi della libreria Quick non verranno gestiti correttamente.

H.4 Utilizzo delle librerie Quick

Questa sezione spiega come caricare una libreria Quick quando si avvia QuickBASIC e come visualizzarne il contenuto. Fornisce inoltre delle indicazioni da seguire se le procedure all'interno di una libreria Quick eseguono calcoli in virgola mobile.

H.4.1 Caricamento di una libreria Quick

Per caricare una libreria Quick, bisogna specificare il nome della libreria richiesta sulla riga di comando al momento dell'avviamento di QuickBASIC, utilizzando la seguente sintassi:

QB [*nomeprogramma*] /L[*nomelibreria*]

Avviando QuickBASIC con l'opzione /L e fornendo il nome di una libreria (*nomelibreria*), QuickBASIC carica la libreria specificata ed entra nell'ambiente di programmazione. Il contenuto della libreria è ora disponibile. Avviando QuickBASIC con l'opzione /L, senza specificare la libreria, esso caricherà la libreria QB.QLB (sezione H.5).

QuickBASIC può essere avviato anche con l'opzione /RUN seguita dal nome del programma (*nomeprogramma*) e dall'opzione /L. In questo caso, QuickBASIC caricherà sia il programma che la libreria Quick specificata, quindi eseguirà il programma senza prima fermarsi nell'ambiente di programmazione.

Nota Quando si utilizzano le librerie Quick per rappresentare i moduli del programma, ricordarsi di aggiornare il file .MAK per mantenerlo conforme ai moduli del programma che si sta sviluppando. (Eseguire ciò per mezzo del comando **Chiudi file** del menu **File**.) Se il file .MAK non viene aggiornato, esso può far caricare a QuickBASIC un modulo contenente una definizione di procedura con un nome uguale a uno già definito nella libreria Quick, il che provoca il messaggio di errore `Duplice definizione`.

Non è possibile caricare più di una libreria Quick alla volta. Specificato un percorso di ricerca, QuickBASIC cerca nel punto indicato; altrimenti, cerca la libreria Quick nei tre punti seguenti:

- La directory corrente
- Il percorso di ricerca delle librerie, specificato dal comando **Percorsi di ricerca**
- Il percorso specificato dalla variabile di ambiente LIB (per informazioni relative alle variabili di ambiente, consultare la propria documentazione DOS)

Esempio

Il comando che segue avvia QuickBASIC ed esegue il programma `PRESENTA.BAS` utilizzando le routine contenute nella libreria `FIG.QLB`:

```
QB /RUN PRESENTA.BAS /L FIG.QLB
```

H.4.2 Aritmetica in virgola mobile in librerie Quick

Le procedure BASIC all'interno delle librerie Quick sono composte di codice compilato dal compilatore della riga del comando BC. Queste procedure condividono con i file eseguibili importanti caratteristiche. Ad esempio, entrambi eseguono calcoli in virgola mobile più rapidamente e con una maggiore precisione rispetto a quelli eseguiti in ambiente QuickBASIC. Per maggiori informazioni, consultare il capitolo 16, "Il menu Esegui", nel manuale *L'ambiente di programmazione Microsoft QuickBASIC*.

H.4.3 Visualizzazione del contenuto di una libreria Quick

Dato che una libreria Quick è essenzialmente un file binario, non è possibile visualizzarne il contenuto con un editor di testo. Il disco originale comprende il programma di utilità QLBDUMP.BAS che permette di elencare tutte le procedure e i simboli dei dati di una determinata libreria. Per visualizzare il contenuto di una libreria Quick, seguire queste indicazioni:

1. Avviare QuickBASIC.
2. Caricare ed eseguire QLBDUMP.BAS.
3. Digitare il nome della libreria Quick da esaminare, senza necessariamente includere l'estensione .QLB.

   Se il file specificato esiste e si tratta di una libreria Quick, il programma visualizza un elenco di tutti i nomi dei simboli della libreria. In questo contesto, essi corrispondono ai nomi delle procedure in essa contenute.

Nel capitolo 3, "I/O su file e su periferica", è contenuto un listato commentato del file QLBDUMP.BAS.

H.5 La libreria fornita (QB.QLB)

Richiamando QuickBASIC con l'opzione /L senza fornire il nome di una libreria Quick, esso caricherà automaticamente la libreria QB.QLB fornita nella confezione QuickBASIC. Questo file contiene tre routine, INTERRUPT, INT86OLD e ABSOLUTE, che forniscono un supporto di interrupt di software per chiamate di sistema e un supporto per l'istruzione **CALL ABSOLUTE**. Per utilizzare le routine di QB.QLB, è necessario che sulla riga di comando venga specificata questa o un'altra libreria nella quale le routine siano state incorporate. Se si desidera utilizzare queste routine insieme ad altre in altre librerie, si faccia una copia della libreria QB.QLB e la si utilizzi come base per creare una libreria contenente tutte le routine desiderate.

H.6 L'estensione .QLB del nome del file

L'estensione .QLB è solo una comoda convenzione. I file libreria Quick, infatti, possono utilizzare qualunque estensione o addirittura nessuna. Tuttavia, nell'elaborazione dell'opzione /L *nomelibreria*, QuickBASIC considera .QLB l'estensione di *nomelibreria*, a meno che non ne sia stata specificata un'altra. Se la libreria Quick è priva di estensione, è necessario far terminare il nome della libreria Quick (*nomelibreria*) con un punto, altrimenti QuickBASIC cercherà un file con il dato nome di base e l'estensione .QLB.

H.7 Creazione di una libreria dalla riga di comando

Dopo aver creato una libreria in ambiente QuickBASIC, si noteranno altri file con estensioni .OBJ e .LIB. Durante la creazione delle librerie Quick, QuickBASIC dirige in realtà il funzionamento di BC, LINK e LIB, e inserisce il risultato delle operazioni sia in una libreria Quick che in una libreria autonoma (.LIB). A processo terminato, esisterà un file oggetto (.OBJ) per ciascun modulo della libreria Quick e un file di libreria (.LIB) contenente un modulo oggetto per ciascun file oggetto. I file con estensione .OBJ sono ora superflui e possono essere eliminati. I file con estensione .LIB sono tuttavia molto importanti e vanno conservati. Queste librerie parallele sono utilizzate da QuickBASIC per creare i file eseguibili dei programmi.

Per creare sia librerie Quick che quelle autonome (.LIB) dalla riga di comando, si possono utilizzare i programmi LINK e LIB in modalità batch. Se si vogliono utilizzare in QuickBASIC routine originariamente scritte e compilate in altri linguaggi, per prima cosa bisogna inserirle in una libreria Quick dalla riga di comando. Una volta fatto ciò, si possono inserire i moduli in BASIC, o dalla riga di comando o dall'interno dell'ambiente QuickBASIC.

I programmatori di software professionale devono accertarsi di fornire ai loro clienti sia le versioni libreria Quick (.QLB) che quelle autonome (.LIB). In assenza di librerie .LIB, gli utenti finali non sarebbero in grado di utilizzare le routine delle librerie nei file eseguibili generati da QuickBASIC.

Quando si crea una libreria Quick utilizzando LINK, la libreria BQLB45.LIB deve essere sempre specificata dopo la terza virgola sulla riga del comando LINK, oppure in risposta al prompt "Librerie".

H.8 Utilizzo di routine in altri linguaggi in una libreria Quick

Per inserire routine scritte in altri linguaggi in una libreria Quick, bisogna che i file oggetto contenenti le routine in altri linguaggi che si vogliono utilizzare siano già compilati o assemblati. A questo scopo si possono usare vari altri linguaggi Microsoft incluso il C, il Microsoft Macro Assembler, il Pascal, il FORTRAN e qualsiasi altro linguaggio che crei file oggetto compatibili con i linguaggi Microsoft.

H.8.1 Costruzione di una libreria Quick

Quanto segue descrive una sessione esempio per la creazione di una libreria Quick contenente routine in altri linguaggi.

1. Si inizia con tre moduli creati rispettivamente in FORTRAN, in C e in Macro Assembler. Prima di tutto si compila o si assembla ciascun modulo con l'interprete del relativo linguaggio, generando i tre file oggetto FOR.OBJ, C.OBJ, e ASM.OBJ.
2. Poi si collegano i file oggetto mediante l'opzione /Q di LINK, che indica al linker di generare un file di libreria Quick, come nella riga seguente:

   ```
   LINK /Q FOR.OBJ C.OBJ ASM.OBJ, MISTI.QLB,,BQLB45.LIB;
   ```

   Il linker interpreta la voce successiva ai nomi dei file oggetto (in questo caso MISTI.QLB) come nome del file risultante dal linkaggio dei moduli. In questo caso, dunque, il file libreria Quick verrà chiamato MISTI.QLB.

3. Ora si crea una libreria parallela .LIB, utilizzando gli stessi file oggetto usati per creare la libreria Quick. In questo caso, il primo nome che segue il comando LIB è quello della libreria .LIB:

   ```
   LIB MISTI.LIB+FOR.OBJ+C.OBJ+ASM.OBJ;
   ```

   E' facile dimenticare questa fase quando si crea una libreria che contiene routine in altri linguaggi, ma è fondamentale se si desidera utilizzare la libreria per creare dei file eseguibili autonomi. Senza queste librerie autonome parallele (.LIB), QuickBASIC non può creare un file eseguibile contenente le loro routine.

4. Con le routine in altri linguaggi ormai presenti in una libreria Quick e i file oggetto originali inclusi in una libreria autonoma avente lo stesso nome di base, si può ora tornare in ambiente QuickBASIC e inserire nella libreria tanti moduli BASIC quanti ne consente la memoria disponibile.

H.8.2 Librerie Quick con zeri iniziali nel primo segmento del codice

Una libreria Quick con zeri iniziali nel primo segmento del codice non è valida, e provoca il messaggio `Errore di caricamento del file` *nomefile* – `Formato non valido`, quando si tenta di caricarla in QuickBASIC. Ciò si verifica, ad esempio, se una routine in linguaggio assembler inserisce nel primo segmento del codice dati inizializzati a zero, e viene successivamente elencata per prima sulla riga del comando LINK durante la creazione di una libreria Quick. In questo caso fare una delle seguenti cose:

- Eseguire il link con un modulo BASIC posto per primo sulla riga del comando LINK.
- Accertare che il primo segmento del codice inizi con un byte diverso da zero, qualunque sia il primo modulo sulla riga del comando LINK.

H.8.3 La routine B_OnExit

QuickBASIC è fornito della funzione BASIC di sistema B_OnExit. Questa funzione può essere utilizzata se delle routine in altri linguaggi eseguono azioni particolari che è necessario annullare prima di una uscita (intenzionalmente o meno) dal programma o prima di una nuova esecuzione. Ad esempio, all'interno dell'ambiente QuickBASIC, un programma in esecuzione che chiama routine in altri linguaggi presenti in una libreria Quick può non terminare sempre normalmente. Se tali routine devono portare a termine operazioni particolari (ad esempio, la disinstallazione di vettori di interrupt) prima di terminare, un richiamo a B_OnExit al loro interno garantirà l'esecuzione di tali operazioni. Di seguito è riportato un esempio di chiamata (per maggiore chiarezza è stato omesso il codice di gestione degli errori). Si noti che questa funzione verrebbe compilata in C in modello ampio.

```
#include <malloc.h>

extern pascal far B_OnExit();      /* Dichiara la routine */

int *p_IntArray;

void InitProc()
{
   void TermProc();      /* Dichiara la funzione TermProc */

   /* Alloca spazio FAR per una matrice a 20 interi:      */
   p_IntArray = (int *)malloc(20*sizeof(int));

   /* Routine di terminazione (TermProc) col BASIC:       */
   B_OnExit(TermProc);
}

void TermProc()          /* La funzione TermProc viene    */
                         /* chiamata prima del riavvio    */
                         /* o del termine del programma.  */
{

   free (p_IntArray);    /* Libera lo spazio FAR allocato */
}                        /* in precedenza da InitProc.    */
```

Se la funzione `InitProc` si trovasse in una libreria Quick, la chiamata B_OnExit assicurerebbe il rilascio dello spazio riservato nella chiamata a malloc, qualora il programma dovesse bloccarsi. La routine può essere chiamata varie volte, dato che il programma potrebbe essere eseguito più volte dall'ambiente QuickBASIC. La funzione `TermProc` stessa, tuttavia, viene chiamata soltanto una volta durante l'esecuzione del programma.

Il seguente brano di programma BASIC è un esempio di chiamata alla funzione `InitProc`.

```
DECLARE SUB InitProc CDECL

X = SETMEM(-2048)       ' Crea spazio per l'allocazione di
                        ' memoria malloc nella funzione C.

CALL InitProc
END
```

Se sono registrate più di 32 routine, B_OnExit restituisce NULL, per indicare che non c'è spazio sufficiente per registrare la routine corrente. (Si noti che B_OnExit ha gli stessi valori di ritorno della routine onexit nella libreria di esecuzione del linguaggio Microsoft C.)

B_OnExit può essere utilizzato con qualunque routine in altro linguaggio (compreso il linguaggio assembler) incluso in una libreria Quick. Il suo utilizzo è facoltativo in programmi compilati o linkati interamente dalla riga di comando.

H.9 Limiti della memoria con le librerie Quick

Dato che una libreria Quick è essenzialmente un file eseguibile (sebbene non sia possibile chiamarla da sola dalla riga di comando DOS), essa occupa più spazio della somma dei suoi file sorgenti. Ciò limita il numero di routine che si possono inserire in una libreria Quick. Per stabilire la dimensione massima consentita a una libreria, sommare la quantità di memoria necessaria al sistema operativo, a QB.EXE ed al modulo principale del programma. Per valutare facilmente questi fattori, riavviare la macchina, quindi avviare QuickBASIC con il programma, e digitare questo comando nella finestra di esecuzione immediata:

```
PRINT FRE (-1)
```

Questo comando visualizza la quantità di memoria disponibile, cioè la dimensione massima di qualsiasi libreria Quick relativa al programma caricato. Nella maggior parte dei casi, la quantità di memoria richiesta per una libreria Quick è più o meno la stessa della dimensione del relativo file sul disco. Fa eccezione a questa regola una libreria con procedure che utilizzano molte stringhe; tale libreria può richiedere una quantità di memoria maggiore.

H.10 Creazione di file eseguibili compatti

Come si è visto precedentemente, QuickBASIC, contemporaneamente ad una libreria Quick, crea una libreria autonoma (.LIB) di moduli oggetto, ciascuno dei quali corrisponde ad un modulo della libreria Quick. Quando si crea un file eseguibile, QuickBASIC cerca nella libreria autonoma (.LIB) i moduli oggetto che contengano le procedure chiamate dal programma.

Se un modulo oggetto nella libreria non contiene procedure chiamate dal programma, esso non sarà inserito nel file eseguibile. Tuttavia, un solo modulo può contenere molte procedure, e se è chiamata dal programma anche una sola di esse, verranno tutte incluse nel file eseguibile. Di conseguenza, anche se un programma utilizza soltanto una delle quattro procedure contenute in un certo modulo, l'intero modulo entrerà a far parte del file eseguibile finale.

Per creare file eseguibili il più compatti possibile, è opportuno dunque creare una libreria in cui ciascun modulo contiene solo procedure strettamente correlate. Se si hanno dei dubbi sul contenuto di una libreria, questo può essere listato dal programma di utilità QLBDUMP.BAS, descritto nella sezione H.4.3, "Visualizzazione del contenuto di una libreria Quick".

Appendice I
Messaggi di errore

Durante la stesura di un programma in QuickBASIC, si possono verificare i seguenti tipi di errori:

- Errori di richiamo
- Errori durante la compilazione
- Errori durante il linkaggio
- Errori durante l'esecuzione

Ciascun tipo di errore è relativo ad una fase particolare del processo di sviluppo del programma. Gli errori di richiamo si verificano quando si richiama QuickBASIC con i comandi QB o BC. Gli errori (e le avvertenze) di compilazione si verificano durante la compilazione, mentre quelli di esecuzione mentre un programma è in esecuzione. Gli errori di linkaggio si verificano solo quando si utilizza il comando LINK per linkare dei file oggetto creati con BC o con compilatori di altri linguaggi.

La sezione I.2 riporta in ordine alfabetico i messaggi degli errori di richiamo, di compilazione e di esecuzione, insieme a ciascun codice di errore ad essi assegnato. La tabella I.1 elenca in ordine numerico i messaggi di errore durante l'esecuzione con i relativi codici di errore. La sezione I.3 presenta i messaggi di errore del Microsoft Overlay Linker, e la sezione I.4 i messaggi di errore del Microsoft Library Manager.

I.1 Visualizzazione dei messaggi di errore

Quando si verifica un errore durante l'esecuzione all'interno dell'ambiente QuickBASIC (con le opzioni schermo predefinite), il messaggio di errore appare in una finestra di dialogo e il cursore viene posizionato sulla riga in cui si è verificato l'errore.

Nei programmi eseguibili autonomi (vale a dire, i programmi che vengono eseguiti mediante la digitazione al prompt di sistema del nome di base del file eseguibile) il sistema di esecuzione visualizza i messaggi di errore seguiti da un indirizzo, a meno che le opzioni /D, /E, o /W non siano state specificate sulla riga del comando BC. In tal caso, al messaggio di errore fa seguito il numero della riga sulla quale l'errore si è verificato. Le forme standard di questo tipo di messaggio di errore sono le seguenti:

`Errore` *n* `nel modulo` *nomemodulo* `all'indirizzo` *segmento: offset*

oppure

`Errore` *n* `nella riga` *numeroriga* `del modulo` *nomemodulo*
`all'indirizzo` *segmento: offset*

Per alcuni errori viene elencato un codice **ERR**. Se si verifica un errore, il valore restituito da **ERR** viene impostato al codice appropriato, all'entrata in una subroutine di gestione degli errori. (L'entrata nelle routine di gestione degli errori avviene per mezzo dell'istruzione **ON ERROR**.) Il valore di **ERR** rimane inalterato finché un'istruzione **RESUME** non restituisce il controllo al programma principale. Per ulteriori informazioni, consultare il capitolo 6, "Gestione degli errori e degli eventi".

La tabella I.1 elenca i codici di errore in ordine numerico. Per le spiegazioni degli errori, consultare l'elenco in ordine alfabetico.

Tabella I.1 Codici degli errori nel corso di esecuzione

Codice	*Descrizione*
2	Errore di sintassi
3	RETURN senza GOSUB
4	Istruzioni DATA esaurite
5	Chiamata di funzione non lecita
6	Overflow
7	Memoria esaurita
9	Indice fuori limite
10	Duplice definizione
11	Divisione per zero
13	Mancata corrispondenza dei tipi

Codice	***Descrizione***
14	Spazio stringhe esaurito
16	Formula a stringa troppo complessa
19	Manca RESUME
20	RESUME senza errore
24	Timeout di periferica
25	Difetto di periferica
27	Carta esaurita
39	Era previsto CASE ELSE
40	E' necessaria una variabile
50	Overflow dell'istruzione FIELD
51	Errore interno
52	Nome o numero di file errato
53	File non trovato
54	Modalità di accesso al file errata
55	File già aperto
56	Istruzione FIELD attiva
57	Errore di I/O su periferica
58	File già esistente
59	Lunghezza del record errata
61	Disco pieno
62	Input oltre la fine del file
63	Numero del record errato
64	Nome del file errato
67	Troppi file
68	Periferica non disponibile
69	Overflow nel buffer di comunicazione
70	Permesso negato
71	Disco non pronto
72	Errore di supporto del disco
73	Caratteristica evoluta non disponibile
74	Riassegnazione di nome tra unità disco diverse
75	Errore di accesso al percorso di ricerca del file
76	Percorso di ricerca non trovato

I.2 Messaggi di errore di richiamo, di errore durante la compilazione, e di errore durante l'esecuzione

`Blocco IF senza END IF`

In una istruzione a blocchi **IF...END IF**, ad **IF** non corrisponde un **END IF** (errore durante la compilazione).

`BYVAL ammette solo argomenti numerici`

BYVAL non accetta argomenti a stringa o a record (errore durante la compilazione).

`/C: Buffer troppo grande`

La dimensione massima del buffer di comunicazione è di 32767 byte (errore di richiamo di BC).

`Carattere di tipo non lecito in una costante numerica`

Una costante numerica contiene un carattere per dichiarazione del tipo non appropriato (errore durante la compilazione).

`Carattere non valido`

QuickBASIC ha trovato un carattere non valido, per esempio un carattere di controllo, nel file sorgente (errore durante la compilazione).

`Caratteristica evoluta non disponibile`

Si sta tentando di utilizzare una caratteristica di QuickBASIC disponibile in un'altra versione del BASIC, o supportata solo in una versione DOS successiva (errore durante la compilazione o durante l'esecuzione).

codice **ERR**: 73

`Carta esaurita`

E' finita la carta della stampante, oppure la stampante è spenta (errore durante l'esecuzione).

codice **ERR**: 27

`CASE senza SELECT`

Manca o è stata scritta in modo errato la prima parte di un'istruzione **SELECT CASE** (errore durante la compilazione).

`Chiamata di funzione non lecita`

Ad una funzione matematica o a stringa è stato passato un parametro fuori intervallo. Un errore di chiamata di funzione si può verificare anche se:

- Viene utilizzato un indice negativo o troppo grande.
- Un numero negativo viene elevato ad una potenza non intera.
- Durante l'utilizzo di **GET** *file* o **PUT** *file*, viene specificato un numero di record negativo.
- Ad una funzione viene assegnato un argomento non valido o fuori intervallo.
- Un'operazione **BLOAD** o **BSAVE** viene diretta ad una periferica non di disco.
- Una funzione o un'istruzione di I/O (ad esempio, **LOC** o **LOF**) viene eseguita su una periferica che non la supporta.
- Una concatenazione di stringhe risulta in una stringa contenente più di 32767 caratteri.

(Errore durante l'esecuzione)

codice **ERR**: 5

`Clausola lecita solo entro un blocco TYPE`

La clausola *elemento* **AS** *tipo* è ammessa solo all'interno di un blocco **TYPE...END TYPE** (errore durante la compilazione).

`COMMON e DECLARE devono precedere le istruzioni eseguibili`

Un'istruzione **COMMON** o **DECLARE** è fuori posto. Queste istruzioni devono precedere qualunque istruzione eseguibile. Tutte le istruzioni BASIC sono eseguibili, tranne le seguenti:

- **COMMON**
- **DEF***tipo*
- **DIM** (per matrici statiche)
- **OPTION BASE**
- **REM**
- **TYPE**
- Tutti i metacomandi

(Errore durante la compilazione)

`COMMON troppo piccolo nella libreria Quick`

Vengono specificate più variabili comuni nel modulo che nella libreria Quick attualmente in memoria (errore durante la compilazione).

`CONST/DIM SHARED segue SUB/FUNCTION`

Le istruzioni **CONST** e **DIM SHARED** vanno anteposte a qualsiasi definizione di sottoprogramma o di procedura **FUNCTION**. Se si sta compilando il programma con BC, questo errore non è "fatale"; il programma eseguirà, sebbene i risultati potranno essere errati (avvertenza durante la compilazione).

`Costante non valida`

E' stata utilizzata un'espressione non valida nell'assegnare un valore ad una costante. Ricordare che le espressioni assegnate alle costanti possono contenere costanti numeriche, costanti simboliche, e qualsiasi degli operatori aritmetici e logici, ad eccezione dell'elevamento a potenza. Un'espressione a stringa assegnata ad una costante deve consistere in un'unica stringa letterale (errore durante la compilazione).

`DECLARE non valida per le procedure del BASIC`

Si è tentato di utilizzare una delle parole chiave **ALIAS**, **CDEL**, o **BYVAL** dell'istruzione **DECLARE** con una procedura del BASIC. **ALIAS**, **CDEL** e **BYVAL** possono essere utilizzate solo con procedure non BASIC (errore durante la compilazione).

`DEF FN non ammessa nelle istruzioni di controllo`

Le definizioni di funzioni **DEF FN** non sono lecite all'interno di costruzioni di controllo quali **IF...THEN...ELSE** o **SELECT CASE** (errore durante la compilazione).

`DEF senza END DEF`

Manca l'istruzione **END DEF** nella definizione di una funzione multiriga (errore durante la compilazione).

`Definizione di funzione nidificata`

La definizione di una **FUNCTION** appare all'interno di un'altra, o all'interno di una clausola **IF...THEN...ELSE** (errore durante la compilazione).

`Deve essere la prima istruzione della riga`

Le istruzioni a blocco **IF...THEN...ELSE**, **IF**, **ELSE**, **ELSEIF**, e **END IF** possono essere precedute solo da un numero o etichetta di riga (errore durante la compilazione).

`Difetto di periferica`

Una periferica ha inviato un messaggio di errore hardware. Se questo messaggio si è avuto durante la trasmissione di dati ad un file di comunicazione, indica che i segnali testati dall'istruzione **OPEN COM** non sono stati trovati nel periodo di tempo specificato (errore durante l'esecuzione).

codice **ERR**: 25

`Digitare il percorso di ricerca del modulo di esecuzione:`

Questo prompt appare se non è stato trovato il modulo di esecuzione BRUN45.EXE. Digitarne la corretta specifica del percorso di ricerca. Si tratta di un errore grave che non può essere gestito (errore durante l'esecuzione).

`Disco non pronto`

Lo sportello dell'unità disco è aperto, oppure non c'è alcun disco nell'unità (errore durante l'esecuzione).

codice **ERR**: 71

`Disco pieno`

Sul disco non c'è spazio sufficiente per portare a termine un'operazione **PRINT**, **WRITE**, o **CLOSE**. Quest'errore può verificarsi anche quando QuickBASIC non ha spazio sufficiente per scrivere un file oggetto o un file eseguibile (errore durante l'esecuzione).

`Divisione per zero`

In un'espressione è stata trovata una divisione per zero o l'elevazione di zero a una potenza negativa (errore durante la compilazione o nel corso dell'esecuzione).

codice **ERR**: 11

`DO senza LOOP`

In un'istruzione **DO...LOOP** manca la clausola conclusiva **LOOP** (errore durante la compilazione).

`Documento troppo grande`

Il documento supera il limite interno del QuickBASIC. E' necessario dividerlo in file separati.

`Dopo /C è previsto il separatore ':'`

Tra l'opzione e l'argomento della dimensione del buffer sono necessari i due punti (errore di richiamo di BC).

`Dopo /C: è prevista la dimensione del buffer`

E' necessario specificare una dimensione di buffer dopo l'opzione /C (errore di richiamo di BC).

`Duplice definizione`

Si è utilizzato un identificatore già definito. Ad esempio, si è tentato di utilizzare lo stesso nome in un'istruzione **CONST** e come definizione di una variabile, oppure lo stesso nome per una procedura e una variabile.

Quest'errore si può verificare anche nel tentativo di ridimensionare una matrice. E' necessario utilizzare **DIM** o **REDIM** per ridimensionare le matrici dinamiche (errore durante la compilazione o durante l'esecuzione).

codice **ERR**: 10

`Duplice etichetta`

Lo stesso numero o la stessa etichetta è stata assegnata a due righe del programma. Uno stesso numero o etichetta può essere assegnato a massimo una riga per modulo (errore durante la compilazione).

`E' necessaria una versione DOS 2.10 o successiva`

Si è tentato di utilizzare QuickBASIC con una versione errata del DOS (errore di richiamo di QB o durante l'esecuzione).

`E' necessaria l'assegnazione di un record/stringa`

In un'istruzione **LSET** manca l'assegnazione della variabile a stringa o a record (errore durante la compilazione).

`E' necessaria l'assegnazione di una stringa`

Ad un'istruzione **RSET** manca l'assegnazione della stringa (errore durante la compilazione).

`E' necessaria un'espressione a stringa`

L'istruzione richiede come argomento un'espressione a stringa (errore durante la compilazione).

`E' necessaria una clausola AS`

Ci si è riferiti senza clausola **AS** ad una variabile dichiarata con tale clausola. Se nella prima dichiarazione della variabile è presente una clausola **AS**, ogni successiva istruzione **DIM**, **REDIM**, **SHARED**, e **COMMON** che si riferisce a quella variabile deve avere una clausola **AS** (errore durante la compilazione).

`E' necessaria una clausola AS alla prima dichiarazione`

Ci si è riferiti con una clausola **AS** ad una variabile dichiarata senza clausola **AS** (errore durante la compilazione).

`E' necessaria una stringa a lunghezza variabile`

In un'istruzione **FIELD** sono ammesse solo stringhe a lunghezza variabile (errore durante la compilazione).

`E' necessaria una variabile`

QuickBASIC ha incontrato un'istruzione **INPUT**, **LET**, **READ**, o **SHARED** senza una variabile come argomento (errore durante la compilazione).

`E' necessaria una variabile`

Un'istruzione **GET** o **PUT** non ha specificato la variabile quando è stata eseguita un'operazione su di un file aperto in modalità **BINARY** (errore durante l'esecuzione).

codice **ERR**: 40

`E' necessaria una variabile a stringa`

L'istruzione richiede come argomento una variabile a stringa (errore durante la compilazione).

`E' necessario DECLARE`

Una chiamata implicita a una procedura **SUB** o **FUNCTION** appare prima della sua definizione. (Una chiamata implicita non utilizza l'istruzione **CALL**.) Le procedure devono essere definite o dichiarate prima di essere implicitamente chiamate (errore durante la compilazione).

`E' necessario un intero tra 1 e 32767`

L'istruzione richiede un intero come argomento (errore durante la compilazione).

`E' prevista una variabile semplice o di matrice`

Il compilatore si aspettava come argomento una variabile (errore durante la compilazione).

`E' previsto GOTO o GOSUB`

QuickBASIC si aspetta un'istruzione **GOTO** o **GOSUB** (errore durante la compilazione).

`E' previsto un identificatore`

Si è tentato di utilizzare un numero o una parola riservata del BASIC al posto di un identificatore (errore durante la compilazione).

`Elemento di matrice dinamica non valido`

Con **VARPTR$** non sono ammessi elementi di matrici dinamiche (errore durante la compilazione).

`Elemento non definito`

Ci si è riferiti a un elemento non definito di un tipo utente. Ad esempio, se il tipo utente `MIOTIPO` contenesse gli elementi A, B, e C, il tentativo di utilizzare la variabile D come elemento di `MIOTIPO` provocherebbe questo messaggio (errore durante la compilazione).

`ELSE senza IF`

Ad una clausola **ELSE** manca la corrispondente clausola **IF**. Talvolta quest'errore viene provocato da istruzioni **IF** nidificate in maniera errata (errore durante la compilazione).

`END DEF senza DEF`

Ad un'istruzione **END DEF** non corrisponde alcuna istruzione **DEF** (errore durante la compilazione).

`END IF senza un blocco IF`

Manca l'inizio di un blocco **IF** (errore durante la compilazione).

`END SELECT senza SELECT`

La prima parte di un'istruzione **SELECT CASE** è mancante o è stata digitata male (errore durante la compilazione).

`END SUB/FUNCTION dev'essere l'ultima riga della finestra`

Si è tentato di aggiungere del codice dopo una procedura. Bisogna ritornare al modulo principale o aprirne un altro (errore durante la compilazione).

`END SUB/FUNCTION senza SUB/FUNCTION`

Un'istruzione **SUB** o **FUNCTION** è stata eliminata per sbaglio (errore durante la compilazione).

`END TYPE senza TYPE`

Un'istruzione **END TYPE** è stata utilizzata al di fuori di una dichiarazione **TYPE** (errore durante la compilazione).

`Era previsto CASE ELSE`

Non è stata trovata un'espressione corrispondente in un'istruzione **SELECT CASE** (errore durante l'esecuzione).

codice **ERR**: 39

`Era previsto(a):` *voce*

Si tratta di un errore di sintassi. Il cursore viene posizionato sulla voce errata (errore durante la compilazione).

`Errata modalità di accesso al file`

Questo errore si verifica nelle seguenti circostanze:

- Quando il programma cerca di utilizzare le istruzioni **PUT** o **GET** in un file ad accesso sequenziale o di eseguire un'istruzione **OPEN** in una modalità file diversa da I, O, o R.
- Quando il programma cerca di utilizzare un'istruzione **FIELD** in un file non aperto per l'accesso casuale.
- Quando il programma cerca di scrivere su un file aperto per input.
- Quando il programma cerca di leggere da un file aperto per output o per accodamento.
- Quando QuickBASIC cerca di utilizzare un file da includere precedentemente memorizzato in formato compresso. I file da includere devono essere memorizzati in formato testo. E' necessario ricaricare il file da includere, memorizzarlo in formato testo e tentare di eseguire nuovamente il programma.
- Quando si cerca di caricare un programma binario alterato.

(Errore durante l'esecuzione)

codice **ERR**: 54

`Errore di $Metacomando`

Un metacomando non è esatto. Se si sta compilando il programma con BC, quest'errore non è "fatale"; il programma potrà essere eseguito, sebbene i risultati possano essere errati (avvertimento durante la compilazione).

`Errore di accesso a un file $INCLUDE`

Il file da includere specificato dal metacomando **$INCLUDE** non è stato trovato (errore durante la compilazione).

`Errore di accesso al percorso di ricerca del file`

Nel corso di un'operazione **OPEN**, **MKDIR**, **CHDIR**, o **RMDIR**, il DOS non è riuscito a collegare esattamente il nome del percorso di ricerca con il nome del file. L'operazione non è stata completata (errore durante la compilazione e durante l'esecuzione).

codice **ERR**: 75

`Errore di caricamento del file (`*nomefile*`) — Errore di I/O su disco`

Quest'errore viene causato da problemi fisici di accesso al disco, ad esempio, se lo sportello dell'unità disco viene lasciato aperto (errore di richiamo di QB).

`Errore di caricamento del file (`*nomefile*`) — Errore nella zona memoria del DOS`

Si è scritto nell'area di memoria utilizzata dal DOS, o con una routine in linguaggio assembler, oppure per mezzo dell'istruzione **POKE** (errore di richiamo di QB).

`Errore di caricamento del file (`*nomefile*`) — Formato non valido`

Si sta tentando di caricare una libreria Quick che non ha il formato giusto. Quest'errore può verificarsi se si tenta di utilizzare una libreria Quick creata con una precedente versione di QuickBASIC, se si sta cercando di utilizzare un file che non è stato elaborato con il comando **Crea libreria** o con l'opzione /QU di LINK, o se si sta cercando di caricare una libreria autonoma (.LIB) con QuickBASIC (errore di richiamo di QB).

`Errore di caricamento del file (`*nomefile*`) — Impossibile trovare il file`

Quest'errore si verifica quando si ridirige l'input da un file a QuickBASIC. Il file di input non si trova nel punto specificato sulla riga di comando (errore di richiamo di QB).

`Errore di caricamento del file (`*nomefile*`) — Memoria esaurita`

E' richiesta più memoria di quella disponibile. Ad esempio, lo spazio memoria potrebbe essere insufficiente ad allocare un buffer di file. Provare a ridurre la dimensione dei buffer DOS, ad eliminare qualunque programma residente in memoria, oppure qualche driver di periferica. Se sono in uso matrici ampie, si può tentare di inserire un meta-comando **$DYNAMIC** all'inizio del programma. La chiusura di eventuali documenti già caricati può liberare dello spazio memoria (errore durante l'esecuzione).

`Errore di I/O su periferica`

Un errore di I/O si è verificato durante un'operazione di I/O su una periferica. Il sistema operativo non può riprendersi dall'errore (errore durante l'esecuzione).

codice **ERR**: 57

`Errore di lettura su input standard`

Si è verificato un errore di sistema durante la lettura dalla console o da un file di input reindirizzato (errore di richiamo di BC).

`Errore di parametro non riconosciuto: "QU"`

Si è tentato di creare un file .EXE o una libreria Quick con una versione errata del Microsoft Overlay Linker. E' necessario utilizzare il linker fornito sui dischi originali QuickBASIC (errore durante la compilazione).

`Errore di sintassi`

Quest'errore può essere causato da diverse condizioni: durante la compilazione, spesso deriva dall'errata digitazione di una parola chiave o argomento del BASIC; durante l'esecuzione, spesso deriva da un'istruzione **DATA** formattata in modo errato (errore durante la compilazione o durante l'esecuzione).

codice **ERR**: 2

`Errore di sintassi in una costante numerica`

Una costante numerica non ha la forma corretta (errore durante la compilazione).

`Errore di sottoprogramma`

Si tratta di un errore di definizione di una **SUB** o **FUNCTION**, di solito provocato perché:

- La **SUB** o la **FUNCTION** è già definita.
- Il programma contiene istruzioni **FUNCTION** o **SUB** nidificate in maniera errata.
- La **SUB** o **FUNCTION** non termina con un'istruzione **END SUB** o **END FUNCTION**.

(Errore durante la compilazione)

`Errore di supporto del disco`

L'hardware dell'unità ha rilevato un difetto fisico sul disco (errore durante l'esecuzione).

codice **ERR**: 72

`Errore durante l'inizializzazione di QuickBASIC`

Le cause di questo errore possono essere varie. Si verifica più comunemente quando non c'è memoria sufficiente per caricare QuickBASIC. Qualora si stia caricando una libreria Quick, provare a ridurre la dimensione della libreria.

Quest'errore si può verificare quando si tenta di utilizzare QuickBASIC su un hardware non supportato (errore di richiamo di QB).

`Errore interno`

Si è verificata una malfunzione interna a QuickBASIC. Compilare il modulo Richiesta di assistenza tecnica incluso nella documentazione, per segnalare alla Microsoft le condizioni che hanno causato il messaggio (errore durante l'esecuzione).

codice **ERR**: 51

`Errore interno presso` *xxxx*

Si è verificata una malfunzione interna a QuickBASIC nel punto *xxxx*. Compilare il modulo Richiesta di assistenza tecnica incluso nella documentazione, per segnalare alla Microsoft le condizioni che hanno causato il messaggio (errore durante la compilazione).

`Errore non visualizzabile`

Non è disponibile un messaggio di errore relativo alla condizione esistente. La causa può essere un'istruzione **ERROR** con un codice di errore non definito (errore durante l'esecuzione).

`Espressione troppo complessa`

Quest'errore viene provocato se si superano certe limitazioni interne. Ad esempio, durante la valutazione di un'espressione, alle stringhe non associate a variabili vengono assegnate posizioni temporanee. Un numero elevato di tali stringhe può causare quest'errore. Cercare di semplificare le espressioni e di assegnare le stringhe a variabili (errore durante la compilazione).

`Etichetta non definita`

Ci si è riferiti (ad esempio in un'istruzione **GOTO**) ad un'etichetta di riga non presente nel programma (errore durante la compilazione).

`Etichetta non definita:` *etichetta*

Un'istruzione **GOTO** *etichettariga* si riferisce ad un'etichetta di riga non esistente (errore durante la compilazione).

`EXIT DO esterno a un ciclo DO...LOOP`

Un'istruzione **EXIT DO** è utilizzata al di fuori di un'istruzione **DO...LOOP** (errore durante la compilazione).

`EXIT esterno a un ciclo FOR...NEXT`

Un'istruzione **EXIT FOR** viene utilizzata al di fuori di un'istruzione **FOR...NEXT** (errore durante la compilazione).

`File da includere troppo grande`

Il file da includere supera il limite interno di QuickBASIC. Scomporlo in diversi file separati (errore durante la compilazione).

`File di input non trovato`

Il file sorgente fornito sulla riga di comando non si trova nella posizione specificata (errore di richiamo di BC).

`File già aperto`

Questo messaggio appare se si è cercato di utilizzare un'istruzione **OPEN** per un file sequenziale già aperto, o un'istruzione **KILL** riferita ad un file aperto (errore durante l'esecuzione).

codice **ERR**: 55

`File già caricato`

Si è tentato di caricare un file già residente in memoria (errore durante la compilazione).

`File già esistente`

Il nome di un file specificato in un'istruzione **NAME** è identico al nome di un file già presente sul disco (errore durante l'esecuzione).

codice **ERR**: 58

`File non trovato`

Un'istruzione **FILES**, **KILL**, **NAME**, **OPEN**, o **RUN** si riferisce ad un file che non esiste (errore durante l'esecuzione).

codice **ERR**: 53

`File non trovato nel modulo` *nome-modulo* `all'indirizzo` *segmento: offset*

Un'istruzione **FILES**, **NAME**, **OPEN** o **RUN** si riferisce ad un file inesistente. Questo messaggio di errore equivale al messaggio `File non trovato`, ma si verifica durante l'esecuzione di programmi compilati. *nome-modulo* è il nome del modulo chiamante. L'indirizzo rappresenta la posizione dell'errore nel codice (errore durante l'esecuzione).

codice **ERR**: 53

`File sorgente ad accesso binario`

Il file che si è cercato di compilare non è un file ASCII. Tutti i file sorgenti salvati con BASICA vanno memorizzati con l'opzione ,A. QuickBASIC utilizza questo messaggio anche come avvertimento se si tenta di utilizzare le opzioni /ZI o /ZD del CodeView con file sorgenti binari (errore durante la compilazione).

`Fine file imprevista in una dichiarazione TYPE`

In un blocco **TYPE...END TYPE** compare un carattere di fine file.

`FOR senza NEXT`

A ciascuna istruzione **FOR** deve corrispondere un'istruzione **NEXT** (errore durante la compilazione).

`Formula a stringa troppo complessa`

Una formula a stringa è troppo lunga, oppure un'istruzione **INPUT** ha richiesto più di 15 variabili a stringa. Scomporre la formula o l'istruzione **INPUT** in più parti per una corretta esecuzione (errore durante l'esecuzione).

codice **ERR**: 16

`Funzione già definita`

Quest'errore si verifica quando viene ridefinita una **FUNCTION** definita precedentemente (errore durante la compilazione).

`Funzione non definita`

E' necessario dichiarare o definire una **FUNCTION** prima di utilizzarla (errore durante la compilazione).

`Guida non trovata. Guida incompleta a causa di errori nel programma`

E' stata richiesta la guida, ma non è stata trovata, e il programma presenta errori che impediscono a QuickBASIC di costruire una tabella di variabili. Premere F5 per visualizzare la riga che ha provocato l'errore.

`Identificatore troppo lungo`

Gli identificatori non devono contenere più di 40 caratteri (errore durante la compilazione).

`Impossibile continuare`

Durante la messa a punto, è stata eseguita una modifica che impedisce la continuazione dell'esecuzione (errore durante l'esecuzione).

`Impossibile generare un listato per file BASIC ad accesso binario`

Si sta tentando di compilare un file sorgente binario con il comando BC e l'opzione /A. Ricompilare senza l'opzione /A (errore di richiamo di BC).

`Impossibile trovare il file (`*nomefile*`). Digitare il percorso:`

Quest'errore si verifica quando QuickBASIC non riesce a trovare una libreria Quick o un programma di utilità (BC.EXE, LINK.EXE, LIB.EXE, o QB.EXE) richiesto dal programma. Digitare il nome esatto del percorso di ricerca, o premere CTRL+C per ritornare al prompt del DOS (errore di richiamo di QB).

`Indice fuori limite`

Si è fatto riferimento ad un elemento di matrice con un indice che superava le dimensioni di questa, oppure si è tentato l'accesso ad un elemento di matrice dinamica non dimensionata. Questo messaggio può essere generato se l'opzione Debug (/D) è stata specificata durante la compilazione. Tale errore si può anche verificare se la dimensione della matrice supera i 64K, la matrice non è dinamica, e non è stata utilizzata l'opzione /AH. In questo caso ridurre la dimensione della matrice, renderla dinamica o utilizzare l'opzione /AH dalla riga di comando (errore durante l'esecuzione).

codice **ERR**: 9

`Input oltre la fine del file`

Un'istruzione **INPUT** ha tentato di leggere da un file nullo (vuoto) o da un file dal quale sono già stati letti tutti i dati. Per evitare questo errore, utilizzare la funzione **EOF** per rilevare la fine del file (errore durante l'esecuzione).

codice **ERR**: 62

`Istruzione FIELD attiva`

Un'istruzione **GET** o **PUT** ha specificato una variabile di record per un file in cui lo spazio era stato allocato dall'istruzione **FIELD**. **GET** o **PUT** con una variabile di record come argomento possono essere utilizzate solo su file sui quali non sono state eseguite istruzioni **FIELD** (errore durante l'esecuzione).

codice **ERR**: 56

`Istruzione ignorata`

Si è utilizzato il comando BC per compilare un programma contenente istruzioni **TRON** e **TROFF**, senza utilizzare l'opzione /D. Questo errore non è "fatale"; il programma eseguirà, sebbene i risultati potrebbero essere errati (avvertenza durante la compilazione).

`Istruzione lecita solo entro una SUB, FUNCTION, o DEF FN`

Quest'istruzione non è lecita nel codice al livello di modulo (errore durante la compilazione).

`Istruzione lecita solo entro una SUB/FUNCTION`

L'istruzione non è lecita nel codice al livello di modulo o nelle funzioni **DEF FN** (errore durante la compilazione).

`Istruzione non lecita in modalità diretta`

L'istruzione è valida solo all'interno di un programma e non può essere utilizzata nella finestra di esecuzione immediata (errore durante la compilazione).

`Istruzione non lecita in un blocco TYPE`

Tra le istruzioni **TYPE** e **END TYPE**, le uniche istruzioni lecite sono **REM** e *elemento* **AS** *nometipo* (errore durante la compilazione).

`Istruzione non lecita in una procedura o una DEF FN`

L'istruzione non è lecita all'interno di una procedura (errore durante la compilazione).

`Istruzione non riconosciuta`

Un'istruzione BASIC è stata, probabilmente, digitata in maniera errata (errore durante la compilazione).

`Istruzione TYPE nidificata in modo errato`

Le definizioni dei tipi definiti dall'utente non sono ammesse nelle procedure (errore durante la compilazione).

`Istruzioni DATA esaurite`

E' stata eseguita un'istruzione **READ** ma nel programma non ci sono più istruzioni **DATA** con dati non ancora letti (errore durante l'esecuzione).

`Istruzioni/etichette non ammesse tra SELECT CASE e CASE`

Le istruzioni e le etichette di riga non sono ammesse tra **SELECT CASE** e la prima istruzione **CASE**. Sono consentiti i commenti ed i separatori di istruzioni (errore durante la compilazione).

`L'istruzione non può precedere le definizioni SUB/FUNCTION`

REM e **DEF***tipo* sono le uniche due istruzioni ammesse prima della definizione di una procedura (errore durante la compilazione).

`L'istruzione non può trovarsi entro un file da includere`

Le istruzioni a blocchi **SUB...END SUB** e **FUNCTION...END FUNCTION** non sono ammesse nei file da includere. Utilizzare il comando **Merge** dal menu **File** per inserire il file da includere nel modulo corrente, oppure caricarlo come modulo separato. In quest'ultimo caso possono risultare necessarie alcune modifiche, perché le variabili condivise rimangono tali solo all'interno di un modulo (errore durante la compilazione).

`L'operazione prevede un disco`

Si è tentato di caricare da o di memorizzare su una periferica non di disco, quale una stampante o la tastiera (errore durante la compilazione).

`Limite inferiore maggiore del limite superiore`

Il limite inferiore definito in un'istruzione **DIM** è maggiore di quello superiore (errore durante la compilazione).

`LOOP senza DO`

Ad un'istruzione **DO...LOOP** manca oppure è stata digitata male la clausola iniziale **DO** (errore durante la compilazione).

`Lunghezza del record errata`

E' stata eseguita un'istruzione **GET** o **PUT** che ha specificato una variabile di record la cui lunghezza non corrisponde alla lunghezza del record specificato nella relativa istruzione **OPEN** (errore durante l'esecuzione).

codice **ERR**: 59

`Manca AS`

Il compilatore richiede una parola chiave **AS**, come in `OPEN "NOMEFILE" FOR INPUT AS #1` (errore durante la compilazione).

`Manca BASE`

QuickBASIC qui prevede l'utilizzo della parola chiave **BASE**, come in **OPTION BASE** (errore durante la compilazione).

`Manca GOSUB`

Ad un'istruzione **ON** *evento* manca l'istruzione **GOSUB** (errore durante la compilazione).

`Manca GOTO`

Ad un'istruzione **ON ERROR** manca l'istruzione **GOTO** (errore durante la compilazione).

`Manca il numero di riga in` *nome-modulo* `all'indirizzo` *segmento: offset*

Quest'errore si verifica se, durante la gestione degli errori, non è possibile trovare l'indirizzo dell'errore nella tabella dei numeri di riga. Ciò accade se non vi sono numeri di riga interi compresi tra 0 e 65527. L'errore si può verificare anche se la tabella dei numeri di riga è stata sovrascritta per sbaglio dal programma dell'utente. Si tratta di un errore grave che non può essere gestito (errore durante l'esecuzione).

`Manca INPUT`

Il compilatore si aspetta la parola chiave **INPUT** (errore durante la compilazione).

`Manca l'opzione On Error (/E)`

Quando si utilizza il comando BC, i programmi che contengono istruzioni **ON ERROR** devono essere compilati con l'opzione On Error (/E) (errore durante la compilazione).

`Manca l'opzione Resume Next (/X)`

Quando si utilizza il comando BC, i programmi che contengono istruzioni **RESUME**, **RESUME NEXT**, o **RESUME 0** devono essere compilati con l'opzione Resume Next (/X) (errore durante la compilazione).

`Manca l'opzione Rilevamento di eventi (/W) o Rilevamento continuo (/V)`

Il programma contiene un'istruzione **ON** *evento* che richiede una di queste opzioni (errore durante la compilazione).

`Manca NEXT per` *variabile*

Ad un'istruzione **FOR** non corrisponde un'istruzione **NEXT**. *variabile* è la variabile di conteggio del ciclo **FOR** (errore durante la compilazione).

`Manca RESUME`

Si è raggiunta la fine del programma nel corso di una routine di gestione degli errori. E' necessaria un'istruzione **RESUME** per ovviare a questo errore (errore durante l'esecuzione).

codice **ERR**: 19

`Manca SUB o FUNCTION`

Ad un'istruzione **DECLARE** non corrisponde alcuna procedura (errore durante la compilazione).

`Manca THEN`

QuickBASIC si aspettava la parola chiave **THEN** (errore durante la compilazione).

`Manca TO`

QuickBASIC si aspettava la parola chiave **TO** (errore durante la compilazione).

`Manca TYPE`

In un'istruzione **END TYPE** manca la parola chiave **TYPE** (errore durante la compilazione).

`Manca un asterisco`

L'asterisco non è presente nella definizione di una stringa in un tipo utente (errore durante la compilazione).

`Manca un punto e virgola`

QuickBASIC si aspettava un punto e virgola (errore durante la compilazione).

`Manca un segno di sottrazione`

QuickBASIC si aspettava un segno di sottrazione (errore durante la compilazione).

`Manca un segno di uguale`

QuickBASIC si aspettava un segno di uguaglianza (errore durante la compilazione).

`Manca un'etichetta o un numero di riga`

Ad un'istruzione, quale **GOTO**, cui è necessaria, manca l'etichetta o il numero di riga (errore durante la compilazione).

`Manca una parentesi destra`

QuickBASIC si aspettava una parentesi destra (conclusiva) (errore durante la compilazione).

`Manca una parentesi sinistra`

QuickBASIC si aspettava una parentesi sinistra, oppure un'istruzione **REDIM** ha cercato di riallocare lo spazio di uno scalare (errore durante la compilazione).

`Manca una virgola`

QuickBASIC qui prevede una virgola (errore durante la compilazione).

`Mancata corrispondenza dei tipi`

La variabile non è del tipo richiesto. Ad esempio, si è cercato di utilizzare l'istruzione **SWAP** con una variabile a stringa e una variabile numerica (errore durante la compilazione o durante l'esecuzione).

codice **ERR**: 13

`Matrice già dimensionata`

Questo errore può essere provocato da:

- Più istruzioni **DIM** per la stessa matrice statica.
- Un'istruzione **DIM** dopo aver utilizzato una matrice. Le matrici dinamiche devono essere disallocate con l'istruzione **ERASE** prima di poter essere ridimensionate con l'istruzione **DIM** o **REDIM**.
- Un'istruzione **OPTION BASE** utilizzata dopo il dimensionamento di una matrice.

(Errore durante la compilazione o nel corso dell'esecuzione)

`Matrice non definita`

Ci si è riferiti ad una matrice non definita (errore durante la compilazione).

`Matrice non dimensionata`

Ci si è riferiti ad una matrice non dimensionata. Se si sta compilando il programma con il comando BC, questo errore non è "fatale"; il programma eseguirà, ma i risultati potranno rivelarsi errati (avvertenza di compilazione).

`Matrice numerica non lecita`

Le matrici numeriche non sono ammesse come argomenti di **VARPTR$**. Sono consentite solo le variabili semplici e gli elementi di matrici a stringa (errore durante la compilazione).

`Matrice troppo grande`

L'utente non dispone di spazio sufficiente per la dichiarazione della matrice. E' necessario, quindi, ridurre la dimensione della matrice o utilizzare il metacomando **$DYNAMIC**. Questo errore può verificarsi anche se la dimensione della matrice supera i 64K, se la matrice non è dinamica, e se non viene utilizzata l'opzione /AH. E' necessario, pertanto, ridurre la dimensione della matrice e utilizzare l'opzione /AH dalla riga di comando (errore durante la compilazione).

`Memoria a distanza alterata`

Lo spazio di memoria riservata a distanza è stato alterato per una delle seguenti ragioni:

- Un programma terminate and stay resident, residente in DOS, non è supportato dal compilatore QB.
- Un'istruzione **POKE** ha modificato le aree di memoria utilizzate da QuickBASIC. (Ciò può modificare il descrittore di una matrice dinamica di numeri o stringhe a lunghezza fissa.)
- Il programma ha chiamato una routine in un altro linguaggio, che ha modificato le aree di memoria utilizzate da QuickBASIC. (Ciò può modificare il descrittore di una matrice dinamica di numeri o stringhe a lunghezza fissa.)

(Errore durante la compilazione)

`Memoria esaurita`

Viene richiesta più memoria di quella disponibile. Ad esempio, può esserci memoria insufficiente per allocare un buffer del file. Si provi a ridurre la dimensione dei buffer del DOS, a eliminare i programmi terminate and stay resident attualmente in memoria, o a eliminare qualche driver di periferica. Se sono in uso matrici molto voluminose, si provi ad inserire un metacomando **$DYNAMIC** all'inizio del programma. La chiusura dei documenti caricati libererà ulteriore spazio memoria (errore di richiamo di BC, durante la compilazione, o durante l'esecuzione).

codice **ERR**: 7

`Modulo non trovato. Eliminare il modulo dal programma?`

Quando il programma è stato caricato, QuickBASIC non ha trovato il file che contiene il modulo indicato. QuickBASIC ha creato al suo posto un modulo vuoto. E' necessario eliminare questo modulo prima di eseguire il programma.

`Nessun modulo principale. Scegliere Esegui Imposta il modulo princ.`

Si è tentato di eseguire il programma dopo aver chiuso il modulo principale. Ciascun programma deve contenere un modulo principale (errore durante la compilazione).

`NEXT senza FOR`

A ciascuna istruzione **NEXT** deve corrispondere un'istruzione **FOR** (errore durante la compilazione).

`Nome COMMON non valido`

QuickBASIC ha incontrato una specifica di /*nomeblocco*/ non lecita (ad esempio, *nomeblocco* uguale a una parola riservata del BASIC) in un blocco **COMMON** con nome (errore durante la compilazione).

`Nome del file errato`

Con un comando **LOAD**, **SAVE**, **KILL** o **OPEN** si è utilizzato un nome di file non lecito (ad esempio, un nome di file con troppi caratteri) (errore durante l'esecuzione).

codice **ERR**: 64

`Nome di file in più ignorato`

Sulla riga di comando sono stati specificati troppi file; i nomi degli ultimi file sulla riga vengono ignorati (errore di richiamo di BC).

`Nome di funzione non valido`

Una parola riservata del BASIC è stata utilizzata come nome di una **FUNCTION** definita dall'utente (errore durante la compilazione).

`Nome di sottoprogramma non valido`

L'errore accade se il nome di un sottoprogramma è una parola riservata del BASIC, oppure se viene utilizzato due volte (errore durante la compilazione).

`Nome di variabile non univoco`

Si è tentato di definire la variabile *x* come tipo utente dopo aver già utilizzato *x.y* (errore durante la compilazione).

`Nome o numero di file errato`

Un'istruzione o un comando si riferisce ad un file con un nome o un numero non specificato nell'istruzione **OPEN** o che sono esterni all'intervallo dei numeri di file precedentemente specificati nel programma (errore durante l'esecuzione).

codice **ERR**: 52

`Non corrisponde il numero degli argomenti`

Si sta utilizzando un numero errato di argomenti in un sottoprogramma o in una funzione BASIC (errore durante la compilazione).

`Non corrispondono i tipi dei parametri`

Un parametro di un sottoprogramma o di una **FUNCTION** non ha lo stesso tipo del corrispondente argomento nell'istruzione **DECLARE** o nella chiamata (errore durante la compilazione).

`Non osservabile`

Quest'errore si verifica durante la specificazione di una variabile da osservare. Assicurarsi che il modulo o la procedura nella finestra di visualizzazione attiva possano accedere alla variabile che si vuole osservare. Ad esempio, il codice a livello di modulo non può accedere a variabili locali ad una **SUB** o **FUNCTION** (errore durante l'esecuzione).

`Non può iniziare per 'FN'`

Sono state utilizzate come prime due lettere del nome di un sottoprogramma o di una variabile le lettere "FN". Esse possono essere utilizzate come prime due lettere solo nella chiamata di una funzione **DEF FN** (errore durante la compilazione).

`Numero del record errato`

In un'istruzione **PUT** o **GET** il numero di un record è minore o uguale a zero (errore durante l'esecuzione).

codice **ERR**: 63

`Numero di dimensioni errato`

Il riferimento ad una matrice contiene un numero errato di dimensioni (errore durante la compilazione).

`Numero non lecito`

Il formato del numero non è valido. Probabilmente si è commesso un errore tipografico. Ad esempio, il numero `2p3` causerà quest'errore (errore durante la compilazione).

`Opzione sconosciuta:` *opzione*

E' stata fornita un'opzione non lecita (errore di richiamo di BC).

`Opzioni valide: [/RUN]`*file*`/AH/B/C:`*buf*`/G/NOHI/H/L[`*lib*`]/MBF/CMD` *str*

Questo messaggio appare quando si richiama il QuickBASIC con un'opzione non valida (errore di richiamo del QuickBASIC).

`Overflow`

Il risultato di un calcolo è troppo elevato per poter essere rappresentato nell'intervallo ammissibile per i numeri in virgola mobile o per i numeri interi (errore durante l'esecuzione).

codice **ERR**: 6

`Overflow dell'istruzione FIELD`

Un'istruzione **FIELD** ha tentato di allocare più byte di quanti erano stati specificati per la lunghezza dei record di un file ad accesso casuale (errore durante l'esecuzione).

codice **ERR**: 50

`Overflow della memoria dati`

Ci sono troppi dati da fissare in memoria. Questo errore è provocato spesso da troppe costanti, o da troppi dati di matrici statiche. Se si sta utilizzando il comando BC, o i comandi **Crea un file EXE** o **Crea libreria**, provare a disattivare le opzioni di messa a punto. Se lo spazio in memoria è ancora insufficiente, scomporre il programma in diverse parti e utilizzare l'istruzione **CHAIN** o il metacomando **$DYNAMIC** (errore durante la compilazione).

`Overflow della memoria programma`

Si è tentato di compilare un programma il cui segmento di codice supera i 64K. Suddividere il programma in più moduli, o utilizzare l'istruzione **CHAIN** (errore durante la compilazione).

`Overflow in una costante numerica`

La costante numerica è troppo grande (errore durante la compilazione).

`Overflow matematico`

Il risultato di un calcolo è troppo grande per essere rappresentato nel formato numerico del BASIC (errore durante la compilazione).

`Overflow nel buffer di comunicazione`

Durante comunicazioni a distanza, è stata superata la capacità del buffer di comunicazione. La dimensione del buffer di comunicazione viene impostata con l'opzione /C dalla riga di comando o dall'opzione **RB** dell'istruzione **OPEN COM**. Provare a controllare più di frequente nel buffer (con la funzione **LOC**) o di svuotarlo più spesso (con la funzione **INPUT$**) (errore durante l'esecuzione).

codice **ERR**: 69

`Parametri formali ripetuti`

La dichiarazione di una **FUNCTION** o di una **SUB** contiene parametri ripetuti, come in questo esempio: `SUB GetName (A,B,C,A) STATIC` (errore durante la compilazione).

`Per ALIAS è necessaria una costante a stringa`

La parola chiave **ALIAS** dell'istruzione **DECLARE** richiede come argomento una costante a stringa (errore di compilazione).

`Percorso di ricerca non trovato`

Durante un'operazione **OPEN**, **MKDIR**, **CHDIR**, o **RMDIR**, il DOS non è riuscito a trovare il percorso di ricerca specificato. L'operazione non è stata completata (errore durante l'esecuzione).

codice **ERR**: 76

`Periferica non disponibile`

La periferica alla quale si è tentato di accedere non è in linea, oppure non esiste (errore durante l'esecuzione).

codice **ERR**: 68

`Permesso negato`

Si è tentato di scrivere su un disco a sola lettura, o di accedere ad un file bloccato (errore durante l'esecuzione).

codice **ERR**: 70

`Procedura già definita nella libreria Quick`

Una procedura della libreria Quick ha lo stesso nome di una procedura del programma (errore durante la compilazione).

`Procedura troppo grande`

La procedura ha superato il limite interno di QuickBASIC. Per rendere utilizzabile la procedura, suddividerla in varie procedure più piccole (errore durante la compilazione).

`Programma troppo esteso a livello di modulo`

Il codice a livello di modulo supera il limite interno di QuickBASIC. Cercare di spostare parte del codice dentro una procedura **SUB** o **FUNCTION** (errore durante la compilazione).

`RESUME senza errore`

Si è incontrata un'istruzione **RESUME** prima di entrare in una routine di gestione degli errori (errore durante l'esecuzione).

codice **ERR**: 20

`RETURN senza GOSUB`

Il programma ha incontrato un'istruzione **RETURN** alla quale non corrisponde una precedente istruzione **GOSUB** libera (errore durante l'esecuzione).

codice **ERR**: 3

`Riassegnazione di nome tra unità disco diverse`

Si è tentato di assegnare un nuovo nome ad un file cambiandone l'unità disco. Ciò non è permesso (prompt durante l'esecuzione).

`Ricominciare da capo`

Si è risposto ad un prompt di **INPUT** con un numero o tipo errato di elementi. Digitare nuovamente la risposta, nella forma corretta (errore durante l'esecuzione).

`Riga non valida. Cominciare da capo`

E' stato utilizzato, dopo uno dei caratteri "\" (barra rovesciata) o ":" (due punti) nel percorso di ricerca, un carattere non valido per il nome di un file (errore di richiamo di BC).

`Riga troppo lunga`

Le righe devono contenere al massimo 255 caratteri (errore durante la compilazione).

`Salto alla successiva istruzione END TYPE`

Un errore nell'istruzione **TYPE** ha fatto ignorare a QuickBASIC tutto ciò che si trova tra le istruzioni **TYPE** e **END TYPE** (errore durante la compilazione).

`SEG o BYVAL non ammesse in CALLS`

BYVAL e **SEG** sono consentite solo in un'istruzione **CALL** (errore durante la compilazione).

`SELECT senza END SELECT`

Manca o è stata digitata male la fine di un'istruzione **SELECT CASE** (errore durante la compilazione).

`Separatore non valido`

C'è un carattere delimitatore non lecito in un'istruzione **PRINT USING** o **WRITE**. Utilizzare un punto e virgola o una virgola come segno delimitatore (errore durante la compilazione).

`Sintassi dell'indice non valida`

Un indice di matrice contiene un errore di sintassi: ad esempio, l'indice contiene sia tipi a stringa che interi (errore durante la compilazione).

`Sono permesse solo variabili semplici`

I tipi utente e le matrici non sono consentiti nelle istruzioni **READ** e **INPUT**. Sono ammesse solo variabili semplici ed elementi di matrici non di tipo utente (errore durante la compilazione).

`Sottoprogramma non definito`

Un sottoprogramma è stato chiamato ma non viene mai definito (errore durante la compilazione).

`Sottoprogrammi non ammessi nelle istruzioni di controllo`

Le definizioni del sottoprogramma **FUNCTION** non sono ammesse all'interno delle costruzioni di controllo, quali **IF...THEN...ELSE** e **SELECT CASE** (errore durante la compilazione).

`Spazio dati esaurito`

Modificare come segue l'utilizzo dello spazio per i dati:

- Utilizzare un buffer di file più piccolo nella clausola **LEN** dell'istruzione **OPEN**.
- Utilizzare il metacomando **$DYNAMIC** per creare matrici dinamiche. I dati di matrici dinamiche possono essere molto più grandi di quelli di matrici statiche.
- Utilizzare matrici di stringhe a lunghezza fissa invece che matrici di stringhe a lunghezza variabile.
- Utilizzare i tipi di dati più compatti possibile. Utilizzare gli interi ovunque possibile.
- Evitare di utilizzare molte procedure piccole. QuickBASIC deve creare diversi byte di informazioni di controllo per ciascuna procedura.
- Usare il comando **CLEAR** per modificare la dimensione dello stack. Riservare solo lo spazio necessario per portare a termine il processo.
- Nel file sorgente non utilizzare righe con più di 256 caratteri, che richiedono ulteriore allocazione di spazio per il buffer di testo.

(Errore durante la compilazione o durante l'esecuzione)

`Spazio stack esaurito`

Quest'errore si può verificare quando una procedura ricorsiva **FUNCTION** viene nidificata troppo in profondità, oppure quando ci sono troppe subroutine, **FUNCTION** e chiamate **SUB** attive. Si può utilizzare l'istruzione **CLEAR** per aumentare lo spazio di stack allocato al programma. Quest'errore non può essere rilevato (errore durante l'esecuzione).

`Spazio stringhe alterato`

Questo errore si verifica quando una stringa non valida viene eliminata dallo spazio stringhe durante la compattazione di memoria. Le cause di questo errore possono essere le seguenti:

- E' stato modificato in modo errato un descrittore di stringa o un puntatore inverso di stringa. Questo può accadere con l'uso di subroutine in linguaggio assembler per la modifica di stringhe.
- Sono stati utilizzati degli indici di matrice fuori intervallo, e lo spazio stringhe è stato inavvertitamente modificato. L'opzione **Genera codice Debug** può essere utilizzata durante la compilazione per verificare se degli indici di matrice ne superano i limiti.
- Un'istruzione **POKE** e/o **DEF SEG** usata male ha modificato lo spazio stringhe in maniera sbagliata.
- Può essersi verificata una mancata corrispondenza tra le dichiarazioni **COMMON** tra due programmi concatenati.

(Errore durante l'esecuzione)

`Spazio stringhe esaurito`

Le variabili a stringa superano la capacità dello spazio destinato alle stringhe (errore durante l'esecuzione).

codice **ERR**: 14

`Specifica del carattere DEF`*xxx* `non valida`

Non è stata digitata correttamente un'istruzione **DEF***tipo*. **DEF** può essere seguita solo da **LNG**, **DBL**, **INT**, **SNG**, **STR**, oppure (per funzioni definite dall'utente) da una spazio vuoto (errore durante la compilazione).

`Specifica del parametro formale non valida`

C'è un errore nell'elenco dei parametri di una funzione o sottoprogramma (errore durante la compilazione).

`STOP nel modulo` *nome* `all'indirizzo` *segmento: offset*

Il programma ha incontrato un'istruzione **STOP** (errore durante l'esecuzione).

`Stringa a lunghezza fissa non lecita`

Si è tentato di utilizzare una stringa a lunghezza fissa come parametro formale (errore durante la compilazione).

`Struttura di controllo in IF...THEN...ELSE incompleta`

In un'istruzione monoriga **IF...THEN...ELSE**, appare un'istruzione **NEXT**, **WEND**, **END IF**, **END SELECT** o **LOOP** senza corrispondente (errore durante la compilazione).

`SUB/FUNCTION senza END SUB/FUNCTION`

Ad una procedura manca l'istruzione finale (errore durante la compilazione).

`Timeout di periferica`

Durante un'operazione di I/O su periferica, il programma non ha ricevuto informazioni nel periodo di tempo specificato (errore durante l'esecuzione).

codice **ERR**: 24

`Tipo non definito`

L'argomento *tipoutente* di un'istruzione **TYPE** non è stato definito (errore durante la compilazione).

`Tipo utente supera i 65535 byte`

Un tipo definito dall'utente non può superare i 64K (errore durante la compilazione).

`Troppe definizioni TYPE`

Il numero massimo consentito di tipi definiti dall'utente è 240 (errore durante la compilazione).

`Troppe dimensioni`

Le matrici vengono limitate a 60 dimensioni (errore durante la compilazione).

`Troppe etichette`

Le righe presenti nell'elenco righe che segue un'istruzione **ON...GOTO** o **ON...GOSUB** sono più di 255 (errore durante la compilazione) o più di 59 (errore durante l'esecuzione in applicazioni compilate).

`Troppe variabili per INPUT`

Un'istruzione **INPUT** è limitata a 60 variabili (errore durante la compilazione).

`Troppe variabili per LINE INPUT`

In un'istruzione **LINE INPUT** è consentita solo una variabile (errore durante la compilazione).

`Troppi argomenti in una chiamata di funzione`

Le chiamate di funzione sono limitate a 60 argomenti (errore durante la compilazione).

`Troppi blocchi COMMON con nome`

Il numero massimo di blocchi **COMMON** con nome è 126 (errore durante la compilazione).

`Troppi file`

Durante la compilazione, quest'errore si verifica se i file da includere sono nidificati su più di 5 livelli; durante l'esecuzione, quando si tenta di creare un nuovo file con un'istruzione **SAVE** o **OPEN** in una directory superando il numero massimo di 255 file (errore durante la compilazione o durante l'esecuzione).

codice **ERR**: 67

`TYPE senza END TYPE`

Ad un'istruzione **TYPE** manca la relativa istruzione **END TYPE** (errore durante la compilazione).

`Un identificatore non può finire per %, &, !, # o $`

Questi suffissi non sono consentiti negli identificatori dei tipi, nei nomi dei sottoprogrammi, e nei nomi che appaiono nelle istruzioni **COMMON** (errore durante la compilazione).

`Un identificatore non può includere un punto`

Né gli identificatori né i nomi degli elementi dei record di tipo utente possono contenere punti. Il punto va utilizzato solo come separatore di variabili di record. Inoltre, un nome di variabile non può contenere un punto se la parte del nome che lo precede è stata utilizzata in una clausola *identificatore* **AS** *tipoutente* in qualunque parte del programma. Se si dispone di programmi che utilizzano il punto nei nomi di variabili, si raccomanda di modificarli perché utilizzino invece lettere maiuscole e minuscole. Ad esempio, cambiare la variabile `ALPHA.BETA` in `AlphaBeta` (errore durante la compilazione).

`Variabile di conteggio FOR già in uso`

Quest'errore si verifica quando una variabile di conteggio viene utilizzata più di una volta in cicli nidificati **FOR** (errore durante la compilazione).

`Variabile di conteggio FOR non valida`

Quest'errore è di solito provocato dall'utilizzo di una variabile di tipo errato come contatore di un ciclo **FOR**. La variabile di conteggio di un ciclo **FOR** deve essere una variabile numerica semplice (errore durante la compilazione).

`Variabili di tipo utente non ammesse nell'espressione`

Le variabili di un tipo definito dall'utente non sono ammesse in espressioni quali `CALL ALPHA((X))`, dove `X` è un tipo utente (errore durante la compilazione).

`WEND senza WHILE`

Questo errore viene provocato quando ad un'istruzione **WEND** non corrisponde un'istruzione **WHILE** (errore durante la compilazione).

`WHILE senza WEND`

Quest'errore viene provocato quando ad un'istruzione **WHILE** non corrisponde un'istruzione **WEND** (errore durante la compilazione).

I.3 Messaggi di errore di LINK

Questa sezione elenca e descrive i messaggi di errore generati dal Microsoft Overlay Linker, LINK.

Gli errori "fatali" costringono il linker a interrompere l'esecuzione, e presentano il formato seguente:

posizione : `errore fatale L1`*xxx* : *testomessaggio*

Gli errori non fatali indicano la presenza di problemi nel file eseguibile. I messaggi di errore non fatale presentano il formato seguente:

posizione : `errore L2`*xxx* : *testomessaggio*

Le avvertenze indicano la presenza di possibili problemi nel file eseguibile, che viene comunque creato da LINK. Le avvertenze presentano il formato seguente:

posizione : `avvertenza L4`*xxx* : *testomessaggio*

In questi messaggi, *posizione* rappresenta il file di input associato all'errore, oppure LINK se non esiste un file di input.

I messaggi di errore che seguono possono apparire quando si esegue il link di file oggetto con LINK:

Numero	***Messaggio di errore di LINK***
L1001	*opzione* : `nome d'opzione ambiguo` Dopo l'indicatore di opzione (/) non è comparso un nome di opzione univoco. Ad esempio, il comando `LINK /N princ;` genera questo errore, perché LINK non può determinare a quale delle tre opzioni inizianti con la lettera "N" ci si riferisce.
L1002	*opzione* : `nome d'opzione non riconosciuto` L'indicatore di opzione (/) era seguito da un carattere non riconosciuto, come nell'esempio seguente: `LINK /ABCDEF princ;`
L1003	`/QUICKLIB, /EXEPACK incompatibili` Sono state specificate due opzioni che non possono essere utilizzate insieme: /QUICKLIB e /EXEPACK.
L1004	*opzione* : `valore numerico non lecito` E' apparso un valore errato di una delle opzioni di LINK. Ad esempio, ad un'opzione che richiede un valore numerico è stata fornita una stringa di caratteri.
L1006	*opzione* : `la dimensione dello stack supera i 65535 byte.` Il valore assegnato come parametro all'opzione /STACKSIZE supera il limite massimo consentito.
L1007	*opzione* : `il numero d'interruzione supera 255` All'opzione /OVERLAYINTERRUPT è stato assegnato un valore nemerico maggiore di 255.
L1008	*opzione* : `limite numero segmenti troppo alto` Utilizzando l'opzione /SEGMENTS il numero di segmenti consentito superava il limite di 3072.

L1009 *numero* : `CPARMAXALLOC : valore non ammesso`

Il numero specificato nell'opzione /CPARMAXALLOC non è compreso nell'intervallo 1-65535.

L1020 `moduli oggetto non specificati`

Nessun nome di file oggetto è stato specificato per LINK.

L1021 `impossibile nidificare file di risposte`

Un file risposte è stato nidificato in un altro.

L1022 `riga di risposta troppo lunga`

Una riga di un file risposte contiene più di 127 caratteri.

L1023 `terminazione da parte dall'utente`

Sono stati digitati i tasti CTRL+C o CTRL+INTERR.

L1024 `parentesi destra nidificata`

Il contenuto di un overlay è stato digitato in maniera errata sulla riga di comando.

L1025 `parentesi sinistra nidificata`

Il contenuto di un overlay è stato digitato in maniera errata sulla riga di comando.

L1026 `manca una parentesi destra`

Alla specifica del contenuto di un overlay sulla riga di comando mancava una parentesi a destra.

L1027 `manca una parentesi sinistra`

Alla specifica del contenuto di un overlay sulla riga di comando mancava una parentesi a sinistra.

L1043 `overflow della tabella di riassegnazione`

Nel programma sono apparse più di 32768 chiamate, salti, o altri puntatori di tipo LONG.

Cercare di sostituire, dove è possibile, riferimenti di tipo LONG con riferimenti a posizione vicina, e poi ricreare il modulo oggetto.

L1045 `troppi record TYPEDEF`

Un modulo oggetto conteneva più di 255 record TYPEDEF. Questi record descrivono variabili comuni. Quest'errore può verificarsi solo nei programmi generati dal compilatore Microsoft FORTRAN o da altri compilatori che supportano variabili comuni (communal). (TYPEDEF è un termine del DOS. Esso viene illustrato nel *Microsoft MS-DOS Programmer's Reference* e in altri volumi di riferimento al DOS.)

L1046 `troppi simboli esterni in un modulo`

In un modulo oggetto sono stati specificati più di 1023 simboli esterni.

Scomporre il modulo in parti più piccole.

L1047 `troppi nomi di gruppo, segmento, e classe in un modulo`

Il programma conteneva troppi nomi di gruppo, di segmento, e di classe.

Ridurre il numero dei gruppi, dei segmenti, e delle classi, e ricreare il file oggetto.

L1048 `troppi segmenti in un modulo`

Un modulo oggetto conteneva più di 255 segmenti.

Suddividere il modulo o unire dei segmenti.

L1049 `troppi segmenti`

Il numero dei segmenti del programma ha superato il limite massimo consentito. Utilizzando l'opzione /SEGMENTS, il cui valore predefinito è 128, si può specificare un più elevato numero massimo consentito di segmenti.

Rieseguire il link utilizzando l'opzione /SEGMENTS con un numero di segmenti adeguato.

L1050 `troppi gruppi in un modulo`

LINK ha incontrato più di ventuno definizioni di gruppi (GRPDEF) in un solo modulo.

Ridurre il numero delle definizioni di gruppi o suddividere il modulo. (Le definizioni dei gruppi vengono illustrate nel *Microsoft MS-DOS Programmer's Reference* e in altri volumi di riferimento al DOS.)

L1051 `troppi gruppi`

Il programma ha definito più di 20 gruppi, senza contare DGROUP.

Ridurre il numero dei gruppi.

`L1052` `troppe librerie`

Si è tentato di eseguire il link con più di 32 librerie.

Unire delle librerie, oppure utilizzare moduli che ne richiedono un numero inferiore.

`L1053` `memoria tabella riferimenti simbolici esaurita`

Non esiste un limite fisso alla dimensione della tabella dei simboli. Tuttavia, la dimensione è limitata dalla quantità di memoria disponibile.

Unire moduli o segmenti e ricreare i file oggetto. Eliminare il maggior numero possibile di simboli pubblici.

`L1054` `limite segmenti richiesto troppo alto`

LINK non disponeva di memoria sufficiente ad allocare tabelle di descrizione per il numero di segmenti richiesto. (Il valore è quello predefinito, 128, oppure quello specificato per mezzo dell'opzione /SEGMENTS.)

Provare ad eseguire nuovamente il link utilizzando l'opzione /SEGMENTS per selezionare un numero inferiore di segmenti (ad esempio, utilizzare 64 se precedentemente era stato usato il valore predefinito), oppure liberare dello spazio memoria eliminando programmi residenti o shell.

`L1056` `troppi overlay`

Il programma ha definito più di 63 overlay.

`L1057` `record di dati troppo grande`

Un record LEDATA (in un modulo oggetto) conteneva più di 1024 byte di dati. Si tratta di un errore del programma traduttore. (LEDATA è un termine DOS, spiegato nel *Microsoft MS-DOS Programmer's Reference* e in altri volumi di riferimento al DOS.)

Notare quale programma traduttore (compilatore o assemblatore) ha prodotto il modulo oggetto errato e in quali circostanze. Si prega di riferire quest'errore alla Microsoft Corporation, mediante il modulo Richiesta di assistenza tecnica incluso nella documentazione.

`L1063` `memoria informazioni CodeView esaurita`

Troppi file oggetto sottoposti al linker contengono informazioni sulla messa a punto. Disattivare l'opzione **Genera codice Debug** nella finestra di dialogo **Crea un file EXE**.

`L1070` `dimensione del segmento superiore a 64K`

Un segmento conteneva da solo più di 64K di codice o di dati.

Cercare di compilare e di eseguire il link utilizzando il modello esteso.

L1071 `il segmento _TEXT supera i 65520 byte`

Quest'errore si verifica soprattutto in programmi C a modello piccolo, ma può accadere con qualsiasi programma contenente un segmento chiamato _TEXT quando viene sottoposto al linker con l'opzione /DOSSEG. I programmi C a modello piccolo devono riservare gli indirizzi di codice 0 e 1; quest'intervallo viene aumentato a 16 per l'allineamento.

L1072 `l'area comune supera i 65536 byte`

Il programma disponeva di più di 64K di variabili comuni (communal). Quest'errore si verifica solo con programmi prodotti dai compilatori che supportano le variabili comuni.

L1080 `impossibile aprire il file di lista`

Il disco o la directory principale è piena.

Eliminare o spostare dei file per creare spazio.

L1081 `spazio insufficiente per il file eseguibile`

Il disco sul quale si stava scrivendo il file eseguibile è pieno.

Liberare dello spazio sul disco e riavviare LINK.

L1083 `impossibile aprire il file eseguibile`

Il disco o la directory principale è piena.

Eliminare o spostare dei file per creare spazio.

L1084 `impossibile creare il file temporaneo`

Il disco o la directory principale è piena.

Liberare dello spazio sul disco e riavviare LINK.

L1085 `impossibile aprire il file temporaneo`

Il disco o la directory principale è piena.

Eliminare o spostare dei file per creare spazio.

L1086 `manca il file di lavoro`

Si è verificato un errore interno.

Comunicare alla Microsoft Corporation le circostanze nelle quali si è verificato l'errore, mediante il modulo Richiesta di assistenza tecnica incluso nella documentazione.

L1087 `fine file imprevista nel file di lavoro`

E' stato rimosso il disco contenente il file temporaneo per l'output di LINK.

L1088 `spazio insufficiente per file di lista`

Il disco su cui si sta scrivendo il file di lista è pieno.

Liberare dello spazio sul disco e riavviare LINK.

L1089 *nomefile* `: impossibile aprire il file risposte`

LINK non è riuscito a trovare il file risposte specificato.

Ciò indica, di solito, un errore di digitazione.

L1090 `impossibile riaprire il file di lista`

Il disco originale non è stato reinserito in seguito al prompt.

Riavviare LINK.

L1091 `fine file imprevista nel file libreria`

Probabilmente è stato rimosso il disco contenente la libreria.

Reinserire il disco contenente la libreria ed eseguire nuovamente LINK.

L1093 `file oggetto non trovato`

Uno dei file oggetto specificati nell'input per LINK non è stato trovato.

Riavviare LINK e specificare il file oggetto.

L1101 `modulo oggetto non valido`

Uno dei moduli oggetto non è risultato valido.

Se l'errore permane dopo una nuova compilazione, si prega di contattare la Microsoft Corporation mediante il modulo Richiesta di assistenza tecnica incluso nella documentazione.

L1102 `fine file imprevista`

E' stato riscontrato un formato non valido per una libreria.

L1103 `tentato accesso a dati fuori dai limiti del segmento`

Un record di dati di un modulo oggetto ha specificato dati che si estendono oltre la fine del segmento. Si tratta di un errore del programma traduttore.

Esaminare quale programma traduttore (compilatore o assemblativo) ha generato il modulo oggetto errato e in quali circostanze. Si prega di riferire questo errore alla Microsoft Corporation mediante il modulo Richiesta di assistenza tecnica incluso nella documentazione.

L1104 *nomefile* `: non è una libreria valida`

Il file specificato non era un valido file di libreria. Questo errore causa l'aborto di LINK.

`L1113` `COMDEF non risolto; errore interno`

Comunicare alla Microsoft Corporation, mediante il modulo Richiesta di assistenza tecnica incluso nella documentazione, le circostanze nelle quali si è verificato l'errore.

`L1114` `file non adatto a /EXEPACK; rieseguire il link omettendolo`

Per il programma sottoposto al linker, la dimensione dell'immagine compattata sommata allo spazio per la compattazione risulta maggiore dell'immagine non compattata. La compattazione è controproducente.

Rieseguire il link senza l'opzione /EXEPACK.

`L1115` `opzione incompatibile con gli overlay`

Sono stati specificati gli overlay ed utilizzata un'opzione (p.es. /QUICKLIB) che non è possibile utilizzare contemporaneamente.

`L2001` `uno o più fixup senza dati`

Un record FIXUPP si è verificato senza essere immediatamente preceduto da un record di dati. Probabilmente si tratta di un errore del compilatore. (Per ulteriori informazioni relative a FIXUPP, si consulti il *Microsoft MS-DOS Programmer's Reference*.)

`L2002` `overflow del fixup a` *posizione* `del segmento` *numero*

Le cause di questo errore possono essere:

- Un gruppo supera i 64K.
- Il programma contiene un salto o chiamata di tipo NEAR da un segmento ad un altro.
- Il nome di un elemento del programma contrasta con quello di una subroutine di libreria inclusa nel linkaggio.
- Una dichiarazione `EXTRN` in un file sorgente in linguaggio assembler è apparsa all'interno di un segmento, come nell'esempio che segue:

```
code     SEGMENT    public 'CODE'
         EXTRN      main:far
start    PROC       far
         call       main
         ret
start    ENDP
code     ENDS
```

E' preferibile la costruzione seguente:

```
EXTRN     main:far
code      SEGMENT    public 'CODE'
start     PROC       far
          call       main
          ret
start     ENDP
code      ENDS
```

Ricontrollare il file sorgente e ricreare il file oggetto. (Per informazioni relative ai segmenti di blocchi e di destinazione, consultare il *Microsoft MS-DOS Programmer's Reference*.)

L2003 `fixup autorelativo intersegmento a` *offset* `nel segmento` *nomesegmento*

Si è utilizzata una chiamata o un salto di tipo NEAR a un elemento di tipo FAR al punto *offset* del segmento *nomesegmento*.

Cambiare la chiamata o il salto al tipo FAR, oppure l'elemento al tipo NEAR.

L2004 `overflow del fixup di tipo LOBYTE`

Un fixup LOBYTE ha generato un overflow di indirizzo. (Per ulteriori informazioni consultare il *Microsoft MS-DOS Programmer's Reference*.)

L2005 `tipo fixup non supportato a` *offset* `nel segmento` *nomesegmento*

Si è verificato un tipo di fixup non supportato dal linker Microsoft. Si tratta probabilmente di un errore del compilatore.

Esaminare le circostanze dell'errore e contattare la Microsoft Corporation utilizzando il modulo di Richiesta di assistenza tecnica incluso nella documentazione.

L2011 *nome* `: incompatibilità NEAR/HUGE`

Ad una variabile comune (communal) sono stati assegnati attributi incompatibili NEAR e HUGE. Questo errore si può verificare solo nei programmi prodotti da compilatori che supportano tali (communal) variabili.

L2012 *nome* `: non corrispondono le dimensioni degli elementi di matrice`

Una matrice comune (communal) di tipo FAR è stata dichiarata con due o più dimensioni differenti per i propri elementi (ad esempio, una matrice è stata dichiarata una volta come matrice di caratteri, ed un'altra come matrice di numeri reali). Quest'errore si verifica solo con compilatori che supportano matrici comuni (communal) di tipo FAR.

`L2013` `record LIDATA troppo grande`

Un record LIDATA contiene più di 512 byte. Questo errore viene di solito provocato da un errore del compilatore.

`L2024` *nome* `: simbolo speciale già definito`

LINK ha riscontrato la ridefinizione di un simbolo pubblico. Rimuovere la/e definizione/i in più.

`L2025` *nome* `: simbolo definito più di una volta`

Rimuovere la definizione in più del simbolo dal file oggetto.

`L2029` `simboli esterni non risolti`

Uno o più simboli sono stati dichiarati esterni in uno o più moduli, ma non sono stati definiti pubblicamente in alcun modulo o libreria. Un elenco dei riferimenti esterni non risolti appare dopo il messaggio, come dimostra l'esempio seguente:

```
simboli esterni non risolti
EXIT in file:
  MAIN.OBJ (main.for)
OPEN in file:
  MAIN.OBJ (main.for)
```

Il nome che precede `in file` è il simbolo esterno non risolto. Sulla riga successiva vengono elencati i moduli oggetto contenenti riferimenti a questo simbolo. Questo messaggio e l'elenco sono scritti anche nel file map, se ne esiste uno.

`L2041` `lo stack più i dati eccedono 64K`

La dimensione totale dei dati di tipo NEAR e dello stack supera i 64K. Ridurre la dimensione dello stack per controllare l'errore.

LINK verifica questa condizione solo se è stata attivata l'opzione /DOSSEG. Questa viene attivata automaticamente dal modulo di avviamento della libreria.

`L2043` `manca il modulo di supporto della libreria Quick`

Il modulo oggetto o una libreria necessaria per creare una libreria Quick non sono stati specificati o non sono stati trovati da LINK. Nel caso di QuickBASIC la libreria fornita è BQLB45.LIB.

L2044 *nome* : `il simbolo ha più definizioni, usare /NOE`

LINK ha riscontrato una possibile ridefinizione di simbolo pubblico. Quest'errore è spesso causato dalla ridefinizione di un simbolo definito in una libreria.

Rieseguire il link utilizzando l'opzione /NOE[XTDICTIONARY].

Quest'errore insieme all'errore `L2025` relativi allo stesso simbolo indica un reale errore di ridefinizione.

L4011 `valore PACKCODE superiore a 65500 non affidabile`

Un segmento Packcode con dimensioni che superano i 65500 byte può non essere affidabile sul processore Intel 80286.

L4012 `l'opzione /HIGH disattiva l'opzione /EXEPACK`

Le opzioni /HIGH e /EXEPACK non possono essere utilizzate contemporaneamente.

L4015 `l'opzione /CODEVIEW disattiva l'opzione /DSALLOCATE`

Le opzioni /CODEVIEW e /DSALLOCATE non possono essere utilizzate contemporaneamente.

L4016 `l'opzione /CODEVIEW disattiva l'opzione /EXEPACK`

Le opzioni /CODEVIEW e /EXEPACK non possono essere utilizzate contemporaneamente.

L4020 *nome* : `la dimensione del segmento di codice supera 65500`

I segmenti di codice di lunghezza compresa tra i 65501 e 65536 byte non sono ammissibili sul processore Intel 80286.

L4021 `nessun segmento di stack`

Il programma non conteneva un segmento di stack definito con il tipo unione `STACK`. Questo messaggio non dovrebbe apparire per i moduli compilati dal Microsoft QuickBASIC, ma può apparire per un modulo in linguaggio assembler.

continua

Normalmente, ogni programma dovrebbe avere un segmento di stack il cui tipo unione sia specificato come `STACK`. E' possibile ignorare questo messaggio se si ha un motivo particolare per non definire uno stack o per definirne uno senza il tipo unione `STACK`. Questo messaggio potrebbe essere provocato dall'esecuzione versioni di LINK anteriori alla 2.40 perché questi linker cercano nelle librerie soltanto una volta.

`L4031` *nome* `: segmento dichiarato in più di un gruppo`

E' stato dichiarato un segmento come facente parte di due gruppi diversi.

Correggere il file sorgente e ricreare i file oggetto.

`L4034` `più di 239 segmenti di overlay; gli eccedenti posti nella radice`

Il programma ha designato più di 239 segmenti per gli overlay. Quando si verifica quest'errore, i segmenti che iniziano col numero 234 vengono collocati nella radice, la parte residente permanente.

`L4045` `il nome del file di output è` *nome*

Il prompt per il campo del file eseguibile ha fornito una predefinizione inesatta, perché l'opzione /QUICKLIB non è stata utilizzata a tempo utile. L'output sarà una libreria Quick con il nome specificato nel messaggio di errore.

`L4050` `troppi simboli pubblici per l'ordinamento`

Il numero dei simboli pubblici supera lo spazio disponibile al loro ordinamento come richiesto dall'opzione /MAP. I simboli non verranno ordinati.

`L4051` *nomefile* `: impossibile trovare la libreria`

LINK non ha trovato il file specificato. Digitare un nuovo nome di file, un nuovo percorso di ricerca, o entrambi.

`L4053` `VM.TMP: nome di file non lecito; ignorato`

VM.TMP è apparso come nome di un file oggetto. Riassegnare il nome al file e rieseguire LINK.

`L4054` *nomefile* `: impossibile trovare il file`

LINK non ha trovato il file specificato. Digitare un nuovo nome di file, un nuovo percorso di ricerca, o entrambi.

I.4 Messaggi di errore di LIB

I messaggi di errore generati dal Microsoft Library Manager, LIB, presentano uno dei seguenti formati:

{*nomefile* | LIB} : errore fatale U1*xxx* : *testomessaggio*

{*nomefile* | LIB} : errore U2*xxx* : *testomessaggio*

{*nomefile* | LIB} : avvertenza U4*xxx* : *testomessaggio*

Il messaggio inizia con il nome del file di input (*nomefile*), se ne esiste uno, o il nome del programma di utilità. Se possibile, LIB visualizza un'avvertenza e continua l'operazione. In alcuni casi gli errori sono gravi, e LIB termina l'elaborazione.

LIB può visualizzare i seguenti messaggi di errore:

Numero	*Messaggio di errore di LIB*
U1150	dimensione pagina troppo piccola La dimensione di pagina di una libreria di input è troppo piccola, il che indica la presenza di un file di input .LIB non valido.
U1151	errore di sintassi: specifica di file non lecita Un operatore di comando, quale il segno meno (-), è stato fornito senza essere seguito da un nome di modulo.
U1152	errore di sintassi: manca il nome dell'opzione Dopo una sbarra (/) non è stata indicata alcuna opzione.
U1153	errore di sintassi: manca il valore dell'opzione All'opzione /PAGESIZE non è stato fornito alcun valore.
U1154	opzione sconosciuta E' stata indicata un'opzione sconosciuta. Attualmente, LIB riconosce solo l'opzione /PAGESIZE.
U1155	errore di sintassi: input non ammesso Il comando specificato non ha rispettato esattamente la sintassi di LIB, come indicato nell'appendice G, "Compilazione ed esecuzione del link da DOS".
U1156	errore di sintassi Il comando specificato non ha rispettato esattamente la sintassi di LIB, come indicato nell'appendice G, "Compilazione ed esecuzione del link da DOS".

`U1157` `manca una virgola o un ritorno a capo`

Una virgola o un ritorno a capo era previsto nella riga di comando, ma mancava. Ciò può indicare che una virgola è stata inserita in maniera errata, come nella riga seguente:

```
LIB MATH.LIB,-MOD1+MOD2;
```

La riga dovrebbe essere stata digitata come segue:

```
LIB MATH.LIB -MOD1+MOD2;
```

`U1158` `manca il carattere di chiusura`

La risposta al prompt "Libreria di output" o l'ultima riga del file risposte utilizzato per avviare LIB non è terminata con un ritorno a capo.

`U1161` `impossibile rinominare la libreria vecchia`

LIB non ha potuto assegnare alla vecchia libreria l'estensione .BAK, poiché ne esisteva già una versione .BAK a sola lettura.

Modificare la protezione della vecchia versione .BAK.

`U1162` `impossibile riaprire la libreria`

Non è stato possibile riaprire la vecchia libreria dopo averle assegnato l'estensione .BAK.

`U1163` `errore nella scrittura al file rimandi interni`

Il disco o la directory principale sono pieni.

Eliminare o spostare dei file per creare spazio.

`U1170` `troppi simboli`

Nel file libreria sono apparsi più di 4609 simboli.

`U1171` `memoria insufficiente`

LIB non ha trovato a disposizione memoria sufficiente all'esecuzione.

Rimuovere qualunque shell o programma residente e ritentare, oppure aggiungere ulteriore memoria.

`U1172` `memoria virtuale esaurita`

Esaminare le circostanze che hanno causato l'errore e contattare la Microsoft Corporation utilizzando il modulo Richiesta di assistenza tecnica incluso nella documentazione.

U1173 `guasto interno`

Esaminare le circostanze che hanno causato l'errore e contattare la Microsoft Corporation utilizzando il modulo Richiesta di assistenza tecnica incluso nella documentazione.

U1174 `marca: memoria non allocata`

Esaminare le circostanze che hanno causato l'errore e contattare la Microsoft Corporation utilizzando il modulo Richiesta di assistenza tecnica incluso nella documentazione.

U1175 `libero: memoria non allocata`

Esaminare le circostanze che hanno causato l'errore e contattare la Microsoft Corporation utilizzando il modulo Richiesta di assistenza tecnica incluso nella documentazione.

U1180 `scrittura al file estrazione fallita`

Il disco o la directory principale sono pieni.

Eliminare o spostare dei file per creare spazio.

U1181 `scrittura al file libreria fallita`

Il disco o la directory principale sono pieni.

Eliminare o spostare dei file per creare spazio.

U1182 *nomefile* `: impossibile creare il file estrazione`

Il disco o la directory principale sono pieni, oppure il file di estrazione è a sola lettura.

Creare spazio sul disco o modificare l'attributo del file.

U1183 `impossibile aprire il file risposte`

Il file risposte non è stato trovato.

U1184 `fine file imprevista nel comando immesso`

Un carattere di fine file è stato ricevuto prematuramente in risposta ad un prompt.

U1185 `impossibile creare la nuova libreria`

Il disco o la directory principale sono pieni, o il file libreria esiste già ed è di tipo a sola lettura.

Creare spazio sul disco o modificare l'attributo del file libreria.

U1186 `errore nella scrittura alla nuova libreria`

Il disco o la directory principale sono pieni.

Eliminare o spostare i file per creare spazio.

U1187 `impossibile aprire VM.TMP`

Il disco o la directory principale sono pieni.

Eliminare o spostare i file per creare spazio.

U1188 `impossibile scrivere a VM`

Esaminare le circostanze che hanno causato l'errore e contattare la Microsoft Corporation utilizzando il modulo Richiesta di assistenza tecnica incluso nella documentazione.

U1189 `impossibile leggere da VM`

Esaminare le circostanze che hanno causato l'errore e contattare la Microsoft Corporation utilizzando il modulo Richiesta di assistenza tecnica incluso nella documentazione.

U1190 `interruzione da parte dell'utente`

L'utente ha premuto i tasti CTRL+C o CTRL+INTERR.

U1200 *nome* `: testata della libreria non valida`

Il file libreria di input presenta un formato non valido. Non è un file libreria, oppure è stato alterato.

U1203 *nome* `: modulo oggetto non valido presso` *posizione*

Il modulo specificato da *nome* non è un modulo oggetto valido.

U2152 *nomefile* `: impossibile creare la lista`

La directory o il disco sono pieni, oppure esiste già un file elenco dei rimandi interni ed è a sola lettura.

Creare spazio sul disco, oppure modificare l'attributo del file.

U2155 *nomemodulo* `: modulo non nella libreria; comando ignorato`

Il modulo specificato non è stato trovato nella libreria di input.

U2157 *nomefile* `: impossibile accedere al file`

LIB non ha potuto aprire il file specificato.

U2158 *nomelibreria* `: testata della libreria non valida; file ignorato`

La libreria di input non ha un formato valido.

U2159 *nomefile* `: formato nonvalido` *numesadec*`; file ignorato`

Il byte di riconoscimento o la parola *numesadec* del file specificato non rientrano tra i tipi riconosciuti: libreria Microsoft, libreria Intel, oggetto Microsoft, o archivio XENIX®.

U4150 *nomemodulo* `: ridefinizione del modulo ignorata`

E' stato specificato un modulo da aggiungere ad una libreria, ma in essa si trovava già un modulo con lo stesso nome. Oppure, un modulo con lo stesso nome è stato trovato più di una volta nella libreria.

U4151 *simbolo* `: simbolo ridefinito nel modulo` *nomemodulo*`, ridefinizione ignorata`

Il simbolo specificato è stato definito in più di un modulo.

U4153 *numero* `: dimensione pagina errata; ignorata`

Il valore specificato nell'opzione /PAGESIZE è minore di 16.

U4155 *nomemodulo* `: modulo non nella libreria`

Nella libreria non è presente il modulo da sostituire. LIB lo aggiunge semplicemente.

U4156 *nomelibreria* `: specifica della libreria di output ignorata`

Oltre al nuovo nome di libreria è stata specificata una libreria di output. Ad esempio, la riga di comando

```
LIB NUOVA.LIB+UN.OBJ,NUOVA.LST,NUOVA.LIB
```

se `NUOVA.LIB` non esiste già, determina questo errore.

U4157 `memoria insufficiente, dizionario esteso non creato`

LIB non ha potuto creare il dizionario esteso. La libreria è ancora valida, ma LINK non può avvalersi di dizionari estesi per eseguire il link più rapidamente.

U4158 `errore interno, dizionario esteso non creato`

LIB non ha potuto creare il dizionario esteso. La libreria è ancora valida ma LINK non può avvalersi di dizionari estesi per eseguire il link più rapidamente.

Indice analitico

P

T